Journal of Machine Learning Research 11 (2010) 2115-2140 Submitted 3/09; Revised 6/10; Published 8/10

# Importance Sampling for Continuous Time Bayesian Networks

**Yu Fan** YFAN@CS.UCR.EDU
**Jing Xu** JINGXU@CS.UCR.EDU
**Christian R. Shelton** CSHELTON@CS.UCR.EDU
*Department of Computer Science and Engineering*
*University of California*
*Riverside, CA, 92521, USA*

**Editor:** Carl Edward Rasmussen

## Abstract

A continuous time Bayesian network (CTBN) uses a structured representation to describe a dynamic system with a finite number of states which evolves in continuous time. Exact inference in a CTBN is often intractable as the state space of the dynamic system grows exponentially with the number of variables. In this paper, we first present an approximate inference algorithm based on importance sampling. We then extend it to continuous-time particle filtering and smoothing algorithms. These three algorithms can estimate the expectation of any function of a trajectory, conditioned on any evidence set constraining the values of subsets of the variables over subsets of the time line. We present experimental results on both synthetic networks and a network learned from a real data set on people's life history events. We show the accuracy as well as the time efficiency of our algorithms, and compare them to other approximate algorithms: expectation propagation and Gibbs sampling.

**Keywords:** continuous time Bayesian networks, importance sampling, approximate inference, filtering, smoothing

## 1. Introduction

Many real world applications involve highly complex dynamic systems. These systems usually contain a large number of stochastic variables, which evolve asynchronously in continuous time. Such dynamic systems include computer networks, sensor networks, social networks, mobile robots, and cellular metabolisms. Modeling, learning and reasoning about these complex dynamic systems is an important task and a great challenge.

### 1.1 Structured Process Representation

A central task of the above applications is to calculate probability distributions of the system over time. For instance, we may wish to know the distribution over when a variable will change next or the state of a current variable, given past (partial) evidence. However, as the number of the variables increases, the state space of the distribution grows exponentially. Such growth makes the inference task very difficult for large systems. One solution is to use structured representation to factorize the state space according to the dependencies of the variables. For dynamic systems, Dynamic Bayesian Networks (DBNs) (Dean and Kanazawa, 1989) are commonly used. A DBN describes the dynamic system as a time-sliced model by measuring the evolution of the system with a (usually fixed) time interval $\Delta t$ . The transition probabilities from states at time $t$ to states at time $t+\Delta t$ are represented by a Bayesian network. DBNs can work well for systems that are observed at regular time steps. However, for many applications, discretizing time has several limitations. First, we usually choose a fixed time interval, $\Delta t$. In many real world systems, variables evolve at different time granularities. Some variables may evolve very fast whereas some evolve very slowly. Choosing an appropriate time interval is a difficult task. Larger $\Delta t$ may result in an inaccurate model while smaller $\Delta t$ may cause inference in the model to be inefficient.

Second, the dependencies of the transition model are unstable with respect to $\Delta t$. That is, different choices of $\Delta t$ may result in different network structures between $t$ and $t+\Delta t$. The network structure represents independencies between variables at $t$ and $\Delta t$. This is a function of $\Delta t$, both theoretically and empirically (Nodelman et al., 2003). If $\Delta t$ is an inherent parameter of the process, this is not a problem. However, if it is chosen for estimation or computational reasons, this becomes an issue as its choice is not unique. Finally, DBNs (and discrete-time Markov processes in general) do not necessarily correspond to processes that are Markovian outside of the sampled instants of time. Consider that if $T$ is the transition matrix for a process with time interval $\Delta t$, $T^{1/2}$ is the transition matrix for the same process with time interval $\frac{\Delta t}{2}$. However, such a square root may not exist in the space of real matrices. Therefore, there may not be any simple extension of a DBN to the times between the sampled instants.

An alternative and more natural approach to model dynamic systems is to use a continuous-time model. For systems with a finite number of states, one way is to consider the entire system as a continuous-time discrete-state Markov process. Like discrete-time processes, this method suffers from the fact that the state space of the process grows exponentially with the number of variables in the system. Recently, Nodelman et al. (2002) extended this framework to a *continuous time Bayesian network* (CTBN), which factorizes a system into local variables using a graphical representation, much as a DBN does for a discrete-time process. Parameter estimation in CTBNs with fully observed data and partially observed data were provided in Nodelman et al. (2003) and Nodelman et al. (2005b) respectively. Because CTBNs explicitly represent the temporal dynamics in continuous time and explore the dependencies among stochastic variables using a structured representation, they have been applied to various real world systems, including human-computer interactions (Nodelman and Horvitz, 2003), server farm failures (Herbrich et al., 2007), robot monitoring (Ng et al., 2005) and network intrusion detection (Xu and Shelton, 2008). Kan and Shelton (2008) used the CTBN representation in their solution of structured continuous-time Markov decision processes.

Queueing theory (Bolch et al., 1998) and Petri nets (Petri, 1962) provide an alternative continuous-time structured process models. However, they make different assumptions about the structure. They were designed to answer questions about steady-state distributions. Their algorithms are not suited to learning from partial data nor to answering many statistical questions. A singular and recent exception is the work of Sutton and Jordan (2008) which applied Gibbs sampling to queueing models.

### 1.2 Prior CTBN Inference Methods

In CTBNs, a trajectory (or sample) consists of the starting values for the system along with the (real-valued) times at which the variables change and their corresponding new values. Inference for a CTBN is the task of estimating the distribution over trajectories given a partial trajectory (in which some values or transitions are missing for some variables during some time intervals). Inference plays a central role as it not only helps us answer queries about distributions, but it is also involved in parameter estimation when the observation data is incomplete. Performing exact inference in a CTBN requires constructing a single rate matrix for the entire system and computing the exponential of the matrix, which is often intractable: the exponentiation must be performed separately for each period of constant evidence and (more problematic) even a sparse representation of the matrix may not fit in memory. Thus, many applications of CTBNs require an approximate inference method. A method based on expectation propagation (Minka, 2001) was presented in Nodelman et al. (2005a). Saria et al. (2007) extended it to full belief propagation and provided a method to adapt the approximation quality.

Other approximate inference methods are based on sampling. They have the advantage of being anytime algorithms. (We can stop at any time during the computation and obtain an answer.) Furthermore, in the limit of infinite samples (computation time), they converge to the true answer.

As we note below, because time is a continuous variable, any evidence containing a record of the change in a variable has a zero probability under the model. Therefore rejection sampling and straightforward likelihood weighting are generally not viable methods.

Ng et al. (2005) developed a continuous-time particle filtering algorithm. However, it only handles point evidence on binary and ternary discrete variables using rejection sampling and focuses primarily on the incorporation of evidence from a continuous-state part of the system (which we do not consider here). Recently, El-Hay et al. (2008) provided another sampling algorithm for CTBNs using Gibbs sampling. The algorithm starts from an arbitrary trajectory that is consistent with the evidence. Then, in each iteration, it randomly picks one variable $X$, and samples an entire trajectory for that variable by fixing the trajectory of all the other variables. Since only $X$ is not fixed, the conditioned cumulative distribution that $X$ stays in one state less than $t$ and the state transition probabilities can be calculated exactly using standard forward and backward propagation within the Markov blanket of $X$. The Gibbs sampling algorithm can handle any type of evidence and it provides an approach to sample from the exact posterior distribution given the evidence. However, the posterior distribution can be any arbitrary function. To sample exactly from it, binary search has to be applied and $F(t)$ is repeatedly evaluated, which may affect the efficiency of the algorithm.

### 1.3 Outline of This Work

In this paper we explore a different sampling approach using importance sampling. Our algorithm generates weighted samples to approximate the expectation of a function of the trajectory. It differs from previous approaches in a number of key ways. There is no exact inference method involved in our approach. Thus, our algorithm does not depend on complex numeric computations. The transition times for variables are sampled from regular exponential distributions in our algorithm, which can be done in constant time. Our algorithm can be adapted to a population-based filter (a particle filter). It can handle both point and continuous evidence, is simple to implement, and can be easily extended to continuous time systems other than CTBNs. The formulation of this sampling procedure is not trivial due to the infinite extent of the trajectory space, both in the transition time continuum and the number of transitions. The algorithm was first presented in Fan and Shelton (2008). This paper extends that work by comparing the algorithm to the newly developed Gibbs sampling algorithm (El-Hay et al., 2008), evaluating its performance on parameter learning with partially observed data, and demonstrating its performance on real-world networks.

The remainder of the paper is structured as follows. In Section 2, we briefly describe the notation for CTBNs. In Section 3, we describe our importance sampling algorithm for CTBNs and extend the algorithm to particle filtering and particle smoothing algorithms. In Section 4, we describe our experiment results.

## 2. Continuous Time Bayesian Networks

We first briefly describe the definition, likelihood, and sufficient statistics of the CTBN model. We then review the exact inference and parameter estimation algorithms for CTBNs.

### 2.1 The CTBN Model

Continuous time Bayesian networks (Nodelman et al., 2002) are based on the framework of continuous time, finite state, homogeneous Markov processes (Norris, 1997). Let $X$ be a continuous time, finite state, homogeneous Markov process with $n$ states $\{x_1,\dots,x_n\}$. The behavior of $X$ is described by the initial distribution $P_X^0$ and the transition model which is often represented by the intensity matrix

$$\mathbf{Q}_X = \begin{bmatrix} -q_{x_1} & q_{x_1x_2} & \cdots & q_{x_1x_n} \\ q_{x_2x_1} & -q_{x_2} & \cdots & q_{x_2x_n} \\ \vdots & \vdots & \ddots & \vdots \\ q_{x_nx_1} & q_{x_nx_2} & \cdots & -q_{x_n} \end{bmatrix},$$

where $q_{x_ix_j}$ is the intensity with which $X$ transitions from $x_i$ to $x_j$ and $q_{x_i} = \sum_{j\neq i} q_{x_ix_j}$. The intensity matrix $\mathbf{Q}_X$ is time invariant. Given $\mathbf{Q}_X$, the transient behavior of $X$ can be described as the following: $X$ stays in state $x_i$ for an amount of time $t$ and transitions to state $x_j$. $t$ is exponentially distributed with parameter $q_{x_i}$. That is, the probability density function and the corresponding distribution function for $X$ staying in state $x_i$

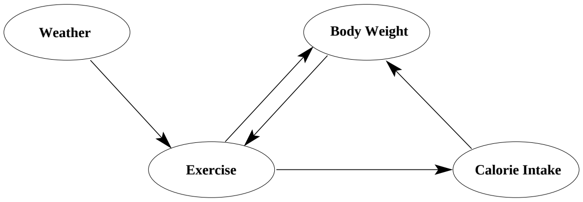


Figure 1: CTBN Example: Weight Control Effect

are

$$f(q_{x_i}, t) = q_{x_i} \exp(-q_{x_i} t), \quad t \geq 0.$$
$$F(q_{x_i}, t) = 1 - \exp(-q_{x_i} t), \quad t \geq 0.$$

The expected time of transitioning is $1/q_{x_i}$. Upon transitioning, the probability $X$ transitions from state $x_i$ to $x_j$ is $\theta_{x_i x_j} = q_{x_i x_j}/q_{x_i}$.The distribution over the state of $X$ at time $t$ can be calculated as

$$P_X(t) = P_X^0 \exp(\mathbf{Q}_X t)$$

where $P_X^0$ is the distribution over $X$ at time 0 represented as a row vector, and exp is the matrix exponential.

To model a dynamic system containing several variables, we can consider the whole system as one variable, enumerate the entire state space, calculate the transition intensity of each pair of these states and put them into a single intensity matrix. However, the size of the state space grows exponentially with the number of variables in the system, which makes this method infeasible for large systems.

Nodelman et al. (2002) defined a *continuous time Bayesian network* (CTBN), which uses a graphical model to provide a compact factored representation of continuous time Markov process. A CTBN models each local variable $X$ as an inhomogeneous Markov process, whose parametrization depends on some subset of other variables $\mathbf{U}$. The intensity matrix of $X$ is called a conditional intensity matrix (*CIM*) $\mathbf{Q}_{X|\mathbf{U}}$, which is defined as a set of intensity matrices $\mathbf{Q}_{X|\mathbf{u}}$, one for each instantiation $\mathbf{u}$ of the variable set $\mathbf{U}$. The evolution of $X$ depends instantaneously on the values of the variables in $\mathbf{U}$.

Let $\boldsymbol{X}$ be a dynamic system containing several variables $X$. A *continuous time Bayesian network* $\mathcal{N}$ over $\boldsymbol{X}$ consists of two components: an *initial distribution* $P_{\boldsymbol{X}}^0$, specified as a Bayesian network $\mathcal{B}$ over $\boldsymbol{X}$, and a *continuous transition model*, specified using a directed (possibly cyclic) graph $\mathcal{G}$ whose nodes are $X \in \boldsymbol{X}$. Let $\mathbf{U}_X$ denote the parents of $X$ in $\mathcal{G}$. Each variable $X \in \boldsymbol{X}$ is associated with a conditional intensity matrix, $\mathbf{Q}_{X|\mathbf{U}_X}$.

**Example 1** *Assume we want to model the behavior of a person controlling his body weight. When the person is overweight, he may exercise more to lose the excess weight. Increasing exercise intensity tends to increase his appetite, which will increase his daily calorie intake. Both exercise intensity and calorie intake contribute to his body weight. Furthermore, the exercise intensity also depends on the weather. Such a dynamic system contains four variables: body weight, exercise, calorie intake, and weather. Each variable changes in continuous time and its change rate depends on the current value of some other variables.*

*We can use a CTBN to represent such behavior. The dependencies of these four variables are depicted using a graphical structure, as shown in Figure 1. The quantitative transient dynamics for each variable is represented using a conditional intensity matrix. Let us assume all the four variables are binary. Let $B(t)$ be the person's body weight ( $Val(B(t)) = \{b_0 = normal, b_1 = overweight\}$), $E(t)$ be the exercise intensity ($Val(E(t) = \{e_0 = light, e_1 = heavy\}$), $C(t)$ be his daily calorie intake ( $Val(C(t) = \{c_0 = low, c_1 = high\}$) and $W(t)$ be the weather ( $Val(W(t) = \{w_0 = rainy, w_1 = sunny\}$). The conditional intensity matrices for the four variables can be specified as*

| | |
|---|---|
| $\mathbf{Q}_W$ | $\mathbf{Q}_W = \begin{bmatrix} -0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix},$ |
| $\mathbf{Q}_{E\mid W,B}$ | $\mathbf{Q}_{E\mid w_0,b_0} = \begin{bmatrix} -0.1 & 0.1 \\ 2 & -2 \end{bmatrix}, \quad \mathbf{Q}_{E\mid w_1,b_0} = \begin{bmatrix} -0.3 & 0.3 \\ 1 & -1 \end{bmatrix},$<br>$\mathbf{Q}_{E\mid w_0,b_1} = \begin{bmatrix} -0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix}, \quad \mathbf{Q}_{E\mid w_1,b_0} = \begin{bmatrix} -1 & 1 \\ 0.1 & -0.1 \end{bmatrix},$ |
| $\mathbf{Q}_{C\mid E}$ | $\mathbf{Q}_{C\mid e_0} = \begin{bmatrix} -0.2 & 0.2 \\ 1 & -1 \end{bmatrix}, \quad \mathbf{Q}_{C\mid e_1} = \begin{bmatrix} -1 & 1 \\ 0.2 & -0.2 \end{bmatrix},$ |
| $\mathbf{Q}_{B\mid E,C}$ | $\mathbf{Q}_{B\mid e_0,c_0} = \begin{bmatrix} -0.2 & 0.2 \\ 0.8 & -0.8 \end{bmatrix}, \quad \mathbf{Q}_{B\mid e_1,c_0} = \begin{bmatrix} -0.1 & 0.1 \\ 1 & -1 \end{bmatrix},$<br>$\mathbf{Q}_{B\mid e_0,c_1} = \begin{bmatrix} -1 & 1 \\ 0.1 & -0.1 \end{bmatrix}, \quad \mathbf{Q}_{B\mid e_1,c_1} = \begin{bmatrix} -0.2 & 0.2 \\ 0.6 & -0.6 \end{bmatrix}.$ |

*Notice that unlike Bayesian networks, the CTBN model allows cycles. The transient behavior of each local variable is controlled by the current value of its parents. If the person is doing light exercise and his calorie intake is low, the dynamics of his body weight are determined by the intensity matrix* $\mathbf{Q}_{B|e_0,c_0}$*. If the time unit is one month, we expect his weight will go back to normal in* $1/0.8 = 1.25$ *months if he is currently overweight and doing light exercise and controlling his daily calorie intake.*

*We can also use a single continuous time Markov process to represent this network, which requires an intensity matrix of size* $16 \times 16$*. To generate the single intensity matrix, we can follow the amalgamation algorithm in Nodelman et al. (2002). Basically, we enumerate the entire state space* $(W,E,C,B)$*, and assign intensity 0 to transitions that change two variables simultaneously. For any transition involving only one of the variables, simply use the entry from the appropriate intensity matrix above. The resulting matrix is*

$$
\begin{array}{l}
w_0e_0c_0b_0 \\ w_1e_0c_0b_0 \\ w_0e_1c_0b_0 \\ w_1e_1c_0b_0 \\ w_0e_0c_1b_0 \\ w_1e_0c_1b_0 \\ w_0e_1c_1b_0 \\ w_1e_1c_1b_0 \\ w_0e_0c_0b_1 \\ w_1e_0c_0b_1 \\ w_0e_1c_0b_1 \\ w_1e_1c_0b_1 \\ w_0e_0c_1b_1 \\ w_1e_0c_1b_1 \\ w_0e_1c_1b_1 \\ w_1e_1c_1b_1
\end{array}
\left[\begin{array}{cccccccccccccccc}
-1 & 0.5 & 0.1 & 0 & 0.2 & 0 & 0 & 0 & 0.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.5 & -1.2 & 0 & 0.3 & 0 & 0.2 & 0 & 0 & 0 & 0.2 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & -3.6 & 0.5 & 0 & 0 & 1 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0.5 & -2.6 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & -2.6 & 0.5 & 0.1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0.5 & -2.8 & 0 & 0.3 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0.2 & 0 & 2 & 0 & -2.9 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0.2 & 0 \\
0 & 0 & 0 & 0.2 & 0 & 1 & 0.5 & -1.9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.2 \\
0.8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0.5 & 0.5 & 0 & 0.2 & 0 & 0 & 0 \\
0 & 0.8 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & -2.5 & 0 & 1 & 0 & 0.2 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & -3 & 0.5 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0.5 & -2.6 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -2.1 & 0.5 & 0.5 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0 & 1 & 0 & 0 & 0.5 & -2.6 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0.6 & 0 & 0 & 0 & 0.2 & 0 & 0.5 & 0 & -1.8 & 0.5 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.6 & 0 & 0 & 0 & 0.2 & 0 & 0.1 & 0.5 & -1.4
\end{array}\right].
$$

*As we include more variables in this system, the size of the intensity matrix grows exponentially with the number of variables.*

### 2.2 Likelihood and Sufficient Statistics

The probability density over trajectories $\sigma$ of a set of variables $\boldsymbol{X}$ described by a CTBN belongs to the exponential family. Therefore, the distribution of a CTBN can be described in terms of the sufficient statistics of $\sigma$ (Nodelman et al., 2003). Let $T[x|\mathbf{u}]$ be the amount of time $X = x$ while $\mathbf{U}_X = \mathbf{u}$, and $M[x, x'|\mathbf{u}]$ be the number of transitions from $x$ to $x'$ while $\mathbf{U}_X = \mathbf{u}$. If we let $M[x|\mathbf{u}] = \sum_{x'} M[x, x'|\mathbf{u}]$, the probability density of trajectory $\sigma$ (omitting the starting distribution) is

$$P_{\mathcal{N}}(\sigma) = \prod_{X \in \boldsymbol{X}} L_X(T[X|\mathbf{U}_X], M[X|\mathbf{U}_X]) \tag{1}$$

where

$$L_X(T[X|\mathbf{U}_X], M[X|\mathbf{U}_X]) = \prod_{\mathbf{u}} \prod_{x} \left( q_{x|\mathbf{u}}^{M[x|\mathbf{u}]} \exp(-q_{x|\mathbf{u}} T[x|\mathbf{u}]) \prod_{x' \neq x} \theta_{xx'|\mathbf{u}}^{M[x,x'|\mathbf{u}]} \right) \tag{2}$$

is the local likelihood for variable $X$. The likelihood also decomposes by time. That is, the likelihood of a trajectory on $[0, T)$ is equal to the likelihood based only on sufficient statistics from time 0 to time $t$ multiplied by the likelihood based only on sufficient statistics from time $t$ to time $T$.

### 2.3 Evidence and Queries

Given a CTBN model, we would like to use it to answer queries conditioned on observations. There are two common types of observations: point evidence and continuous evidence. Point evidence represents the observation of the value of some variables at a particular time instant. Continuous evidence provides the behavior of some variables throughout an interval $[t_1, t_2)$. For instance, $x = 1$ during the interval $[2, 3.5)$, or $x = 1$ from $t = 2$ to $t = 3$ and then $x$ transitions to $x = 0$ at $t = 3$ and stays in that state until $t = 5$. We define $x[t_1 : t_2)$ be the behavior of variable $X$ on the interval $[t_1, t_2)$, $x[t_1 : t_2]$ be the behavior of $X$ on the interval $[t_1, t_2]$ and $x(t_1 : t_2]$ be the behavior of $X$ on the interval $(t_1, t_2]$.

Queries can ask about the marginal distribution of some variables at a particular time, such as the distribution of $x$ and $y$ at $t = 2$, or questions about the timing of a transition, such as the distribution over the time that $y$ transitions from $y = 1$ to $y = 2$ for the first time in the interval $[1, 4)$. In learning (especially when employing expectation-maximization), we might query the expected sufficient statistics of a CTBN, which include the total amount of time that a variable spends in a state, and the total number of times that a variable transitions from one state to another state under certain conditions. For example, we might want to know the total amount of time that $x = 0$ throughout the entire interval, or the number of times that $x$ transitions from 1 to 2 during the time interval $[2, 3)$ when $y = 0$. In this paper, we will concentrate on answering queries given the continuous evidence, but our method can be trivially extended to point evidence.

### 2.4 Exact Inference in CTBNs

A CTBN can be viewed as a homogeneous Markov process with a large joint intensity matrix amalgamated from the CIMs of the CTBN. Exact inference in a CTBN can be performed by generating a single joint intensity matrix over the entire state space of the CTBN and running the forward-backward algorithm on the joint intensity matrix of the homogeneous Markov process. We review this method here, but a more complete treatment can be found in Nodelman et al. (2002).

Assume that we have a partially observed trajectory $\sigma$ of a CTBN $\mathcal{N}$ from 0 to $T$. We can divide the evidence $\sigma$ into $N$ intervals $[t_i, t_{i+1})$ $(i = 0, \ldots, N-1)$ according to the observed transition times. That is, each interval contains a constant observation of the CTBN, and $t_i$ is the time that a variable begins to be observed, stops being observed, or is observed to transition. We set $t_0 = 0$ and $t_N = T$.

To perform exact inference, we first generate the intensity matrix $\mathbf{Q}$ for the joint homogeneous Markov process and incorporate the evidence into $\mathbf{Q}$. If each variable $X_i$ in the CTBN $\mathcal{N}$ has $n_i$ states, the number of states of the joint Markov process is $n = \prod n_i$ and $\mathbf{Q}$ is an $n \times n$ matrix. The value of the off-diagonal element $q_{ij}$ in $\mathbf{Q}$ for which only one variable value is different between states $i$ and $j$ is the corresponding intensity in the CIM of that variable. All the other off-diagonal elements are zero since two variables can not transition at exactly the same time in a CTBN. The diagonal elements are computed to make each row sum to zero.

To incorporate the evidence, we reduce the joint intensity matrix $\mathbf{Q}$ to $\mathbf{Q}_i$ for each interval $[t_i, t_{i+1})$ by zeroing out the rows and columns of $\mathbf{Q}$ which represent states that are inconsistent with the evidence. Additionally, let $\mathbf{Q}_{i,j}$ be the matrix $\mathbf{Q}$ with all elements zeroed out except the off-diagonal elements that represent the intensities of transitioning from non-zero rows in $\mathbf{Q}_i$ to non-zero columns in $\mathbf{Q}_j$. If evidence blocks $i$ and $j$ differs only in which variables are observed (no transition is observed between them), then $\mathbf{Q}_{i,j}$ is the identity matrix instead.

$\exp(\mathbf{Q}_i(t_{i+1}-t_i))$ represents the transition matrix for interval $[t_i,t_{i+1})$ and $\mathbf{Q}_{i,i+1}$ corresponds to the transition probability density between two consecutive intervals at time $t_{i+1}$. We can use the forward-backward algorithm for Markov process to answer queries.

We define the forward and backward probability vectors $\boldsymbol{\alpha}_t$ and $\boldsymbol{\beta}_t$ as

$$\begin{aligned}\boldsymbol{\alpha}_t &= p(X_t,\sigma_{[0,t)}),\\ \boldsymbol{\beta}_t &= p(\sigma_{[t,T)}|X_t)\ .\end{aligned}$$

Let $\boldsymbol{\alpha}_0$ be the initial distribution $P_X^0$ over the state and $\boldsymbol{\beta}_T$ be a vector of ones. The forward and backward distribution vector for each interval can be calculated recursively:

$$\begin{aligned}\boldsymbol{\alpha}_{t_{i+1}} &= \boldsymbol{\alpha}_{t_i}\exp(\mathbf{Q}_i(t_{i+1}-t_i))\mathbf{Q}_{i,i+1},\\ \boldsymbol{\beta}_{t_i} &= \mathbf{Q}_{i-1,i}\exp(\mathbf{Q}_i(t_{i+1}-t_i))\boldsymbol{\beta}_{t_{i+1}}\ .\end{aligned}$$

The distribution over the state of the CTBN at time $t\in[t_i,t_{i+1})$ given the evidence $\sigma_{[0,T)}$ can be computed as

$$P(X_t=k,\sigma_{[0,T)})=\boldsymbol{\alpha}_{t_i}\exp(\mathbf{Q}_i(t-t_i))\boldsymbol{\Delta}_{k,k}\exp(\mathbf{Q}_i(t_{i+1}-t))\boldsymbol{\beta}_{t_{i+1}} \tag{3}$$

where $\boldsymbol{\Delta}_{i,j}$ is an $n\times n$ matrix of zeros with a single one in position $i,j$. Other queries can be similarly computed.

### 2.5 CTBN Parameter Estimation

Given a set of trajectories $D=\{\sigma_1,\sigma_2,\ldots,\sigma_n\}$ and a fixed graphical structure, we would like to estimate the parameters (the conditional intensity matrix) of the CTBN model.

When the data set $D$ is complete, where each trajectory $\sigma_i$ is a complete set of state transitions and the times at which they occurred, the parameters can be learned by maximizing the log-likelihood of the data set (Nodelman et al., 2003). According to Equation 1 and Equation 2, the log-likelihood can be written as the sum of the log-likelihood for each local variable. By maximizing the log-likelihoods, the parameters can be derived as

$$\hat{q}_{x|\mathbf{u}}=\frac{M[x|\mathbf{u}]}{T[x|\mathbf{u}]}\,;\quad \hat{\theta}_{xx'|\mathbf{u}}=\frac{M[x,x'|\mathbf{u}]}{M[x|\mathbf{u}]}\ . \tag{4}$$

When the data set is incomplete, the expectation maximization (EM) algorithm (Dempster et al., 1977) can be used to find the maximum likelihood parameters (Nodelman et al., 2005b). The EM algorithm begins with an arbitrary initial parameter assignment, and alternatively repeats the expectation step and maximization step until convergence. In expectation step, for each trajectory $\sigma_i\in D$, expected sufficient statistics $\bar{M}[x|\mathbf{u}]$, $\bar{M}[x,x'|\mathbf{u}]$ and $\bar{T}[x|\mathbf{u}]$ are computed using exact inference. In maximization step, new parameters are computed according to Equation 4 as if the expected sufficient statistics came from complete data.

## 3. Sampling-based Inference

As we described in the previous section, exact inference in a CTBN can be performed by generating a single joint intensity matrix over the entire state space. As the number of states is exponential in the number of the nodes in the network, this approach is infeasible when the network size is large. In this section we describe an algorithm for approximate CTBN inference based on importance sampling.

```
Procedure CTBN-Sample($t_{end}$)
  1. $t \leftarrow 0$, $\sigma \leftarrow \emptyset$
  2. For each variable $X \in \boldsymbol{X}$
         Choose state $x(0)$ according to $\boldsymbol{\theta}^{\mathcal{B}}_{X|\mathbf{pa}_{\mathcal{B}}(X)}$.
  Loop:
  3. For each variable $X$ such that $Time(X)$ is undefined:
       Choose $\Delta t$ for next $X$ transition from an exponential with parameter $q_{x(t)|\mathbf{u}_X(t)}$.
       Define $Time(X) \leftarrow t + \Delta t$
  4. Let $X = \arg\min_{X \in \boldsymbol{X}} [Time(X)]$
  5. If $Time(X) \geq t_{end}$ return $\sigma$
  6. Update $t \leftarrow Time(X)$
  7. Choose $x(t)$, the next value of $X$, from the multinomial with parameters $\boldsymbol{\theta}_{x(t)|\mathbf{u}_X(t)}$.
       Add $\langle X \leftarrow x(t), t \rangle$ to $\sigma$.
       Undefine $Time(X)$, and $Time(Y)$ for all variables $Y$ for which $X \in \mathbf{U}_Y$.
```

Figure 2: Forward sampling semantics for a CTBN

### 3.1 Forward Sampling

Queries that are not conditioned on evidence can be answered by randomly sampling many trajectories and looking at the fraction that match the query. More formally, if we have a CTBN $\mathcal{N}$ we generate a set of particles $\mathcal{D} = \{\sigma[1], \ldots, \sigma[M]\}$ where each particle is a sampled trajectory. With $\mathcal{D}$ we can estimate the expectation of any function $g$ by computing

$$\hat{\mathbf{E}}_{\mathcal{N}}[g] = \frac{1}{M} \sum_{m=1}^{M} g(\sigma[m]) \ . \tag{5}$$

For example, if we let $g = \mathbf{1}\{x(5) = x_1\}$ then we could use the above formula to estimate $P_{\mathcal{N}}(x(5) = x_1)$. Or the function $g(\sigma)$ might count the total number of times that $X$ transitions from $x_1$ to $x_2$ while its parent $U$ has value $u_1$, allowing us to estimate the expected sufficient statistic $M[x_1, x_2|u_1]$. The algorithm for sampling a trajectory is shown in Figure 2. For each variable $X \in \boldsymbol{X}$, it maintains $x(t)$—the state of $X$ at time $t$—and $Time(X)$—the next potential transition time for $X$. The algorithm adds transitions one at a time, advancing $t$ to the next earliest variable transition. When a variable $X$ (or one of its parents) undergoes a transition, $Time(X)$ is resampled from the new exponential waiting time distribution. We use $\mathbf{u}_X(t)$ to represent the instantiation to parents of $X$ at time $t$.

If we want to obtain a conditional probability of a query given evidence, the situation is more complicated. We might try to use *rejection sampling*: forward sample to generate possible trajectories, and then simply reject the ones that are inconsistent with our evidence. The remaining trajectories are sampled from the posterior distribution given the evidence, and can be used to estimate probabilities as in Equation 5. However, this approach is entirely impractical in our setting, as in any setting involving an observation of a continuous quantity—in our case, time. In particular, suppose we observe that $X$ transitions from $x_1$ to $x_2$ at time $t$. The probability of sampling a trajectory in which that transition occurs at precisely that time is zero. Thus, if we have evidence about transitions, with probability 1, none of our sampled trajectories will be relevant.

### 3.2 Gibbs Sampling

Recently, El-Hay et al. (2008) provided a Markov Chain Monte Carlo (MCMC) procedure which used a Gibbs sampler to generate samples from the posterior distribution given the evidence.

Suppose we want to sample trajectories from a CTBN with $n$ variables $(X_i, X_2, \ldots, X_n)$ given the evidence $\mathbf{e}$. The Gibbs sampler starts with an arbitrary trajectory that is consistent with the evidence. In each iteration, the sampler randomly picks one variable $X_i$ and samples the entire trajectory of $X_i$ by fixing the trajectories of the other variables $Y = \{X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n\}$ as evidence. To generate the entire trajectory of $X_i$ according to the evidence **e**, the states and transitions of $X_i$ need to be sampled in those intervals that $X_i$ is not observed according to the evidence. The trajectory in each unobserved interval of $X_i$ can be generated by alternatively sampling transition time $\Delta t$ and new state $x$ from the posterior distribution given **e** and the trajectories of the other variables $Y$.

Assume we are sampling the trajectory of $X$ for the interval $[0,T]$, and $X_i(0) = x_0$, $X_i(T) = x_T$. The transition time $\Delta t$ is sampled by inverse transform sampling: first draw $\xi$ from the $[0,1]$ uniform distribution and set $\Delta t = F^{-1}(\xi)$, where $F^{-1}(\xi)$ is the inverse of the conditional cumulative distribution function $F(t)$ that $X_i$ stays in state $x_0$ for a time less than $t$:

$$F(t) = 1 - Pr(X_i(0:t] = x_0 | x_0, x_T, Y[0:T]) \; .$$

$F(t)$ can be calculated by decomposing $Pr(X_i(0:t] = x_0|x_0, x_T, Y[0:T])$ using the Markov property of the process:

$$Pr(X_i(0:t] = x_0|x_0, x_T, Y[0:T]) = \frac{\tilde{\alpha}(t)\tilde{\beta}_{x_0}(t)}{\tilde{\beta}_{x_0}(0)}$$

where

$$\tilde{\alpha}(t) = Pr(X_i(0:t] = x_0, Y[0:t]|x_0, Y_0),$$
$$\tilde{\beta}_x(t) = Pr(x_T, Y(t:T]|X_i(t) = x, Y(t)) \; .$$

$\tilde{\alpha}(t)$ and $\tilde{\beta}_x(t)$ can be calculated using a slightly modified version of the standard forward-backward algorithm described in Section 2.4. Using the fact that $X_i$ is independent of all the other components given the entire trajectory of its Markov blanket, the computation of $\tilde{\alpha}(t)$ and $\tilde{\beta}(t)$ can be limited to $X_i$ and its Markov blanket (the parents of $X_i$, the children of $X_i$, and the children's parents).

Since the conditional cumulative distribution function $F(t)$ can be arbitrarily complex, the inverse function $F^{-1}(t)$ can not be solved analytically. Finding $\Delta t$ that satisfies $F(\Delta t) = \xi$ is performed using a two-step searching method: first find the interval $[\tau_k, \tau_{k+1}]$ that satisfies $F(\tau_k) < \xi < F(\tau_{k+1})$, where $\tau_k$ are the transition points of the Markov blanket of $X_i$. Then $\Delta t$ is found by performing an $L$ step binary search on the interval $[\tau_k, \tau_{k+1}]$.

The transition probability that $X_i$ transitions from $x^{(0)}$ to a new state $x$ can be calculated similarly:

$$Pr(X_i(t^+) = x|X_i(0:t] = x^{(0)}, Y(0:T]) = \frac{q^{X_i|Y}_{x_0,x}\tilde{\beta}_x(t)}{\sum_{x' \neq x_0} q^{X_i|Y}_{x_0,x'}\tilde{\beta}_{x'}(t)} \; .$$

The Gibbs sampling algorithm can handle any type of evidence. The sampled trajectories are guaranteed to be consistent with the evidence. However, sampling the transition time $\Delta t$ requires using a binary search algorithm and repeatedly computing the conditional cumulative distribution function $F(t)$, which may require long running time.

### 3.3 Importance Sampling

In this section, we introduce another approximate inference method using importance sampling, which does not require computing the exact posterior distribution. This method first appeared in Fan and Shelton (2008).

In importance sampling, we generate samples from a proposal distribution $P'$ which guarantees that our sampled trajectories will conform to our evidence **e**. We must weight our samples to correct for the fact that we are drawing them from $P'$ instead of the target distribution $P_{\mathcal{N}}$ defined by the CTBN. In particular, if $\sigma$ is a sample from $P'$ we set its weight to be

$$w(\sigma) = \frac{P_{\mathcal{N}}(\sigma, \mathbf{e})}{P'(\sigma)} \; . \qquad (6)$$

In normalized importance sampling, we draw a set of samples $\mathcal{D} = \{\sigma[1], \ldots, \sigma[M]\}$ i.i.d. from the proposal distribution, and estimate the conditional expectation of a function $g$ given evidence $\mathbf{e}$ as

$$\hat{\mathbf{E}}_{\mathcal{N}}[g \mid \mathbf{e}] = \frac{1}{W} \sum_{m=1}^{M} g(\sigma[m]) w(\sigma[m])$$

where $W$ is the sum of the weights.

This estimator is consistent if the support of $P'$ is a superset of the support of $P_{\mathcal{N}}$. In general, $\hat{\mathbf{E}}_{\mathcal{N}}$ is biased and the bias decreases as $O(M^{-1})$. The variance of the estimator also decreases as $O(M^{-1})$. For more information on this and related sampling estimates, see Hesterberg (1995).

For our algorithm, we base the proposal distribution on the forward sampling algorithm. As we are sampling a trajectory, we occasionally depart from the regular forward sampling algorithm and "force" the behavior of one or more variables to ensure consistency with the evidence.

### 3.4 Simple Evidence

The simplest query involves evidence over some subset of variables $\boldsymbol{V} \subset \boldsymbol{X}$ for the total length of the trajectory. We force only the behavior of the variables $\boldsymbol{V}$ and there are no choices about how to do that. In particular, we use the following proposal distribution: forward sample the behavior of variables $X \in (\boldsymbol{X} \setminus \boldsymbol{V})$ inserting the known transitions at known times for variables in $\boldsymbol{V}$ as determined by the evidence. As there are no choices in our forcing, the likelihood of drawing $\sigma$ from the proposal distribution is just the likelihood contribution of forward sampling the behavior of the variables $X \in (\boldsymbol{X} \setminus \boldsymbol{V})$, in the context of the total behavior of the system.

According to Section 2.2, $x[t_1 : t_2)$ can be summarized by the sufficient statistics over $X$ on the interval $[t_1, t_2)$. Let $\tilde{L}_X(x[t_1 : t_2))$ be a partial likelihood contribution function, computed by plugging the sufficient statistics of $x[t_1 : t_2)$ into Equation 2. The partial contribution function can be defined over a collection of intervals $I$ as $\tilde{L}_X(I) = \prod_{x[t_1 : t_2) \in I} \tilde{L}_X(x[t_1 : t_2))$. Returning to our simple evidence above, let $\tau_1 < \tau_2 \ldots, \tau_{n-1} < \tau_n$ be all the transition times in $\sigma_{[0,T)}$, $\tau_0 = 0$ and $\tau_{n+1} = T$. The likelihood of drawing $\sigma$ from the target distribution $P_{\mathcal{N}}$ is

$$\tilde{L}_{\mathcal{N}}(\sigma) = \prod_{X \in \boldsymbol{X}} \prod_{i=0}^{n} \tilde{L}_X(x[\tau_i : \tau_{i+1}))$$

Let $\tilde{L}'_X(x[t_1 : t_2))$ be the corresponding probability density for our sampling procedure. Since we force the values and transitions of variables in $\boldsymbol{V}$ according to the evidence, the probability that we sample an interval $x[\tau_i : \tau_{i+1})$ for $X \in \boldsymbol{V}$ from proposal distribution $P'$ is always 1. Therefore, the likelihood of drawing $\sigma$ from the proposal distribution $P'$ is

$$\begin{aligned}\tilde{L}'_{\mathcal{N}}(\sigma) &= \prod_{X \in \boldsymbol{X}} \prod_{i=0}^{n} \tilde{L}'_X(x[\tau_i : \tau_{i+1})) \\ &= \prod_{X \in (\boldsymbol{X} \setminus \boldsymbol{V})} \prod_{i=0}^{n} \tilde{L}_X(x[\tau_i : \tau_{i+1})) \times \prod_{X \in \boldsymbol{V}} \prod_{i=0}^{n} 1.\end{aligned}$$

To compute the proper weight $w(\sigma)$ we substitute in Equation 6, and get

$$\begin{aligned}w(\sigma) = \frac{P_{\mathcal{N}}(\sigma, \mathbf{e})}{P'(\sigma)} &= \frac{\prod_{X \in \boldsymbol{X}} \prod_{i=0}^{n} \tilde{L}_X(x[\tau_i : \tau_{i+1}))}{\prod_{X \in (\boldsymbol{X} \setminus \boldsymbol{V})} \prod_{i=0}^{n} \tilde{L}_X(x[\tau_i : \tau_{i+1}))} \\ &= \prod_{X \in \boldsymbol{V}} \prod_{i=0}^{n} \tilde{L}_X(x[\tau_i : \tau_{i+1})).\end{aligned}$$

Therefore, the weight $w(\sigma)$ is the likelihood contribution of all the variables in $\boldsymbol{V}$. This algorithm exactly corresponds to *likelihood weighting* in Bayesian networks (Shachter and Peot, 1989; Fung and Chang, 1989).

Intuitively, this makes sense because we can account for all the evidence by simply assigning the observed trajectories to the observed variables.

### 3.5 General Evidence

Now, consider a general evidence pattern **e**, in which we have time instants where variables become observed or unobserved. How can we force our trajectory to be consistent with **e**? Suppose there is a set of variables which has evidence beginning at $t_e$. We can not simply force a transition at time $t_e$ to make the variables consistent with the evidence **e**: if the set contains more than one variable, the sample would have multiple simultaneous transitions, an event whose likelihood is zero.

Instead, we look ahead for each variable we sample. If the current state of the variable does not agree with the upcoming evidence, we force the next sampled transition time to fall before the time of the conflicting evidence. To do this, we sample from a truncated exponential distribution instead of the full exponential distribution. In particular, if we are currently at time $t$ and there is conflicting evidence for $X$ at time $t_e > t$, we sample from an exponential distribution with the same $q$ value as the normal sampling procedure, but where the sample for $\Delta t$ (the time to the next transition) is required to be less than $t_e - t$. The probability density of sampling $\Delta t$ from this truncated exponential is $\frac{q\exp(-q\Delta t)}{1-\exp(-q(t_e-t))}$ where $q$ is the relevant intensity for the current state of $X$ (the diagonal element of $\mathbf{Q}_{X|\mathbf{U}_X}$ corresponding to the current state of $X$).

The subsequent state is still sampled from the same (forward sampling) distribution. In Section 3.6 we explore a more intelligence option. Note that we cannot, in general, transition directly to the evidence state, as such a transition may not be possible (have 0 rate). Furthermore, if we are still "far away" from the upcoming evidence, such a transition may lead to a highly unlikely trajectory resulting in an inefficient algorithm.

To calculate the weight $w(\sigma)$, we partition $\sigma$ into two pieces. Let $\sigma_e$ be the collection for all variables $X \in \boldsymbol{X}$ of intervals $x[t_1 : t_2)$ where the behavior of $X$ is set by the evidence. Let $\sigma_s$ be the complement of $\sigma_e$ containing the collection of intervals of unobserved behavior for all variables. By applying Equation 6, we have

$$\begin{aligned} w(\sigma) &= \frac{P_{\mathcal{N}}(\sigma, \mathbf{e})}{P'(\sigma)} \\ &= \prod_{x[\tau_i : \tau_{i+1}) \in \sigma_s} \frac{\tilde{L}_X(x[\tau_i : \tau_{i+1}))}{\tilde{L}'_X(x[\tau_i : \tau_{i+1}))} \times \prod_{x[\tau_i : \tau_{i+1}) \in \sigma_e} \frac{\tilde{L}_X(x[\tau_i : \tau_{i+1}))}{\tilde{L}'_X(x[\tau_i : \tau_{i+1}))} \\ &= \prod_{x[\tau_i : \tau_{i+1}) \in \sigma_s} \frac{\tilde{L}_X(x[\tau_i : \tau_{i+1}))}{\tilde{L}'_X(x[\tau_i : \tau_{i+1}))} \times \prod_{x[\tau_i : \tau_{i+1}) \in \sigma_e} \tilde{L}_X(x[\tau_i : \tau_{i+1})). \end{aligned} \tag{7}$$

Based on the distribution we sampled for transition time of the variable in each step, we can further partition $\sigma_s$ into three pieces:

> $\sigma_{sn}$ be the collection for all variables $X \in \boldsymbol{X}$ of intervals $x[t_1 : t_2)$ where the transition time is sampled from an exponential distribution.
>
> $\sigma_{st}$ be the collection for all variables $X \in \boldsymbol{X}$ of intervals $x[t_1 : t_2)$ where the transition time is sampled from a truncated exponential distribution and the variable is involved in the next transition.
>
> $\sigma_{sf}$ be the collection for all variables $X \in \boldsymbol{X}$ of intervals $x[t_1 : t_2)$ where the transition time is sampled from a truncated exponential distribution and the variable is not involved in the next transition.

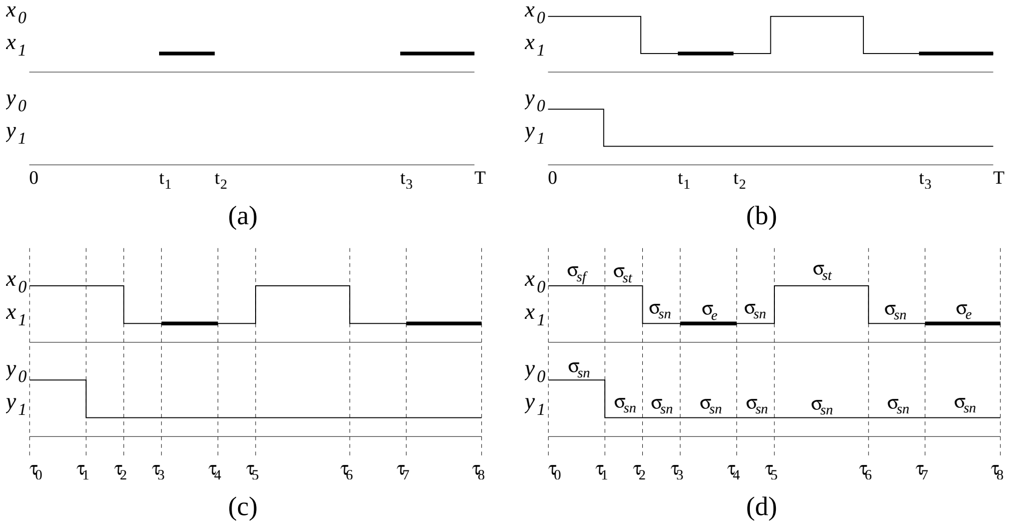

(a)
(b)
(c)
(d)
$x_0$
$x_1$
$y_0$
$y_1$
0
$t_1$
$t_2$
$t_3$
T
$\tau_0$ $\tau_1$ $\tau_2$ $\tau_3$ $\tau_4$ $\tau_5$ $\tau_6$ $\tau_7$ $\tau_8$
$\sigma_{sf}$ $\sigma_{st}$ $\sigma_{sn}$ $\sigma_e$ $\sigma_{sn}$ $\sigma_{st}$ $\sigma_{sn}$ $\sigma_e$
$\sigma_{sn}$
$\sigma_{sn}$ $\sigma_{sn}$ $\sigma_{sn}$ $\sigma_{sn}$ $\sigma_{sn}$ $\sigma_{sn}$ $\sigma_{sn}$

Figure 3: (a) Evidence of a CTBN. (b) A sampled trajectory agreeing with the evidence. (c). Partitioning of the trajectory according to the evidence and the transitions. $\sigma_e$ equals $x[\tau_3:\tau_4)$ and $x[\tau_7:\tau_8)$ (d) Partitioning of the trajectory based on the different sampling situations.

Therefore, we can rewrite Equation 7 as

$$w(\sigma) = \prod_{x[\tau_i:\tau_{i+1})\in\sigma_{sn}} \frac{\tilde{L}_X(x[\tau_i:\tau_{i+1}))}{\tilde{L}'_X(x[\tau_i:\tau_{i+1}))} \times \prod_{x[\tau_i:\tau_{i+1})\in\sigma_{st}} \frac{\tilde{L}_X(x[\tau_i:\tau_{i+1}))}{\tilde{L}'_X(x[\tau_i:\tau_{i+1}))}$$
$$\times \prod_{x[\tau_i:\tau_{i+1})\in\sigma_{sf}} \frac{\tilde{L}_X(x[\tau_i:\tau_{i+1}))}{\tilde{L}'_X(x[\tau_i:\tau_{i+1}))} \times \prod_{x[\tau_i:\tau_{i+1})\in\sigma_e} \tilde{L}_X(x[\tau_i:\tau_{i+1})). \quad (8)$$

**Example 2** *Assume that we are given a CTBN with two binary variables $X$ and $Y$. $X$ has two states $x_0$ and $x_1$. $Y$ has two states $y_0$ and $y_1$. We have such observation: $X$ is $x_1$ in interval $[t_1,t_2)$ and $[t_3,T)$, as shown in Figure 3(a). To answer queries based on the evidence, we use the method above to sample trajectories. Figure 3(b) shows one of the sampled trajectories. To calculate the weight of the trajectory, we partition the trajectory into four categories (as shown in Figure 3(c) and Figure 3(d)), and apply Equation 8.*

According to Equation 8, each time we add a new transition to the trajectory, we advance time from $t$ to $t+\Delta t$. For each variable $x$ we must update the weight of trajectory to reflect the likelihood ratio for $x[t:t+\Delta t]$ based on the distribution we use to sample the "next time" and the transition variable we select. Each such variable can be considered separately as their times are sampled independently.

For any variable $x$ whose value is given in the evidence during the interval $[t,t+\Delta t)$, as we discussed above, the contribution to the trajectory weight is just $\tilde{L}_{\mathcal{N}}(x[t:t+\Delta t))$. For any variable $x[t:t+\Delta t)\in\sigma_{ns}$, whose "next time" was sampled from an exponential distribution, $\tilde{L}_X(x[\tau_i:\tau_{i+1}))=\tilde{L}'_X(x[\tau_i:\tau_{i+1}))$ and the ratio is 1.

Now, we consider segments $x[t:t+\Delta t)\in\sigma_{st}$ and $x[t:t+\Delta t)\in\sigma_{sf}$. The behavior of the variables in these segments are forced due to upcoming evidence.

For variable $X$ that $x[t:t+\Delta t)\in\sigma_{st}$, the variable's "next time" is sampled from a truncated exponential distribution and it is part of the next transition. The weight must be multiplied by the probability density of sampling the transition in $P_{\mathcal{N}}$ divided by the the probability density in the sampling algorithm. The former is an exponential distribution and the latter is the same exponential distribution, truncated to be less than $t_e - t$. The ratio of these two probabilities is $1 - \exp(-q(t_e - t))$, where $q$ is the relevant intensity.

Otherwise, $x[t : t + \Delta t) \in \sigma_{sf}$, the next time for the variable was sampled from a truncated exponential but was longer than $\Delta t$. In this case, the ratio of the probabilities of a sample being greater than $\Delta t$ is $\frac{1-\exp(-q(t_e-t))}{1-\exp(-q(t_e-t-\Delta t))}$. Note that when $\Delta t$ is small (relative to $t_e - t$, the time to the next evidence point for this variable), the ratio is almost 1. So, while the trajectory's weight is multiplied by this ratio for every transition for every variable that does not agree with the evidence, it does not overly reduce the weight of the entire trajectory.

The algorithm for CTBN importance sampling is shown in Figure 4. To more easily describe the evidence, we define a few helper functions:

$\mathbf{e}_X^{val}(t)$ is the value of $X$ at time $t$ according to the evidence, or undefined if $X$ has no evidence at $t$.

$\mathbf{e}_X^{time}(t)$ is the first time after $t$ when $\mathbf{e}_X^{val}(t)$ is defined.

$\mathbf{e}_X^{end}(t)$ is the first time after or equal to $t$ when $\mathbf{e}_X^{val}(t)$ changes value or becomes undefined.

Note that $\mathbf{e}_X^{end}(t) = t$ when there is point evidence at $t$, when $t$ is the end of an interval of evidence, and when there is a transition in the evidence at time $t$.

The line numbers follow those given in the forward sampling algorithm with new or changed lines marked with an asterisk. *Time*$(X)$ might be set to the end of an interval of evidence which is not a transition time but simply a time when we need to resample a next potential transition. This means that we will not update $\sigma$ with a new transition every time through the loop. The algorithm differs from the forward sampling procedure as follows. Step 2 now accounts for evidence at the beginning of the trajectory (using standard likelihood weighting for Bayesian networks). In Step 3, we draw $\Delta t$ from the truncated exponential if the current value disagrees with upcoming evidence. If the current evidence includes this variable, $\Delta t$ is set to the duration of such evidence. Step 5 updates the weights using the procedure *Update-Weight*. Finally, Step 7 now deals with variables that are just leaving the evidence set.

### 3.6 Predictive Lookahead

The algorithm in Figure 4 draws the next state for a variable from the same distribution as the forward sampling algorithm. This may cause a variable to transition several times in a short interval before evidence as the variable "searches" to find a way to transition into the evidence. Thus, we may generate many unlikely samples, making the algorithm inefficient. We can help mitigate this problem by trying to force the variable into a state that will lead to the evidence.

When sampling the next state for variable $X$ at time $t$, instead of sampling from the multinomial according to $\boldsymbol{\theta}_{x(t)|\mathbf{u}_X(t)}$, we would like to sample from the distribution of the next state conditioned on the upcoming evidence. Suppose $X$ is in state $x_i$ at time $t$, and the next evidence for $X$ is state $x_k$ at $t_e$. Assuming the parents of $X$ do not change before $t_e$ and ignoring evidence over the children of $X$, the distribution of the state of $X$ at $t$ given only the evidence can be calculated using Equation 3:

$$\tilde{P}(X_{t+\Delta t} = x_j | X[t : t + \Delta t) = x_i, X_{t_e} = x_k) = \frac{1}{Z}\mathbf{1}_j^\top \mathbf{Q}_X \exp(\mathbf{Q}_X(t_e - t))\mathbf{1}_k = p_{i,j}$$

where $\mathbf{1}_j$ is the vector of zeros, except for a one in position $j$. We can therefore select our new state according to the distribution of $\tilde{P}(X_{t+\Delta t}|X[t : t+\Delta t) = x_i, X_{t_e} = x_k)$ and, assuming state $x_j$ is selected, multiply the weight by $\frac{\theta_{x_i x_j|\mathbf{u}_X(t)}}{p_{i,j}}$ to account for the difference between the target and sampling distributions.

### 3.7 Particle Filtering

The algorithm in Figure 4 allows us to generate a single trajectory and its weight, given the evidence. To apply this algorithm to the task of online inference in a dynamic system, we can generate multiple trajectories in parallel, advancing time forward as evidence is obtained.

```
Procedure CTBN-Importance-Sample($t_{end}, \mathbf{e}$)
  1. $t \leftarrow 0$, $\sigma \leftarrow \emptyset$, $w \leftarrow 1$                                                  *
  2. For each variable $X \in \boldsymbol{X}$
         If $\mathbf{e}_X^{val}(0)$ defined,
             set $x(0) \leftarrow \mathbf{e}_X^{val}(0)$,                                                   *
             Set $w \leftarrow w \cdot \theta^{\mathcal{B}}_{x(0)|\mathbf{pa}_{\mathcal{B}}(0)}$                                               *
         Else
             choose state $x(0)$ according to $\boldsymbol{\theta}^{\mathcal{B}}_{X|\mathbf{pa}_{\mathcal{B}}(X)}$
  Loop:
  3. For each $X \in \boldsymbol{X}$ such that $Time(X)$ is undefined:
         If $\mathbf{e}_X^{val}(t)$ is defined, set $\Delta t \leftarrow \mathbf{e}_X^{end}(t) - t$                          *
         Elseif $\mathbf{e}_X^{val}(t_e)$ is defined where $t_e = \mathbf{e}_X^{time}(t)$, $x(t) \neq \mathbf{e}_X^{val}(t_e)$,
             choose $\Delta t$ from an exponential distribution with
             parameter $q_{x(t)|\mathbf{u}_X(t)}$ given $\Delta t < (t_e - t)$.                                *
         Else choose $\Delta t$ from an exponential w/ param. $q_{x(t)|\mathbf{u}_X(t)}$
         Define $Time(X) \leftarrow t + \Delta t$
  4. Let $X = \arg\min_{X \in \boldsymbol{X}} [Time(X)]$
  5. If $Time(X) \geq t_{end}$
             $w \leftarrow$ Update-Weight($X, w, t, t_{end}$)                                             *
             return $(\sigma, w)$
     Else                                                                                     *
             $w \leftarrow$ Update-Weight($X, w, t, Time(X)$)                                          *
  6. Update $t \leftarrow Time(X)$
  7. If $\mathbf{e}_X^{end}(t) \neq t$ or $\mathbf{e}_X^{val}(t)$ is defined                                              *
         If $\mathbf{e}_X^{val}(t)$ is defined, set $x(t) \leftarrow \mathbf{e}_X^{val}(t)$                                  *
         Else choose $x(t)$, the next value of $X$, from a
             multinomial with parameter $\boldsymbol{\theta}_{x(t)|\mathbf{u}_X(t)}$
         Add $\langle X \leftarrow x(t), t \rangle$ to $\sigma$.
         Undefine $Time(X)$ and $Time(Y)$ for all variables $Y$
         for which $X \in \mathbf{U}_Y$
     Else                                                                                     *
         Undefine $Time(X)$.                                                                  *

  Procedure Update-Weight($Y, w, t_1, t_2$)
  1. For each $X \in \boldsymbol{X}$ such that $\mathbf{e}_X^{val}(t)$ is defined for $t \in [t_1, t_2)$:
       $w \leftarrow w \cdot \tilde{L}_X(x[t_1 : t_2))$
  2. For each $X \in \boldsymbol{X}$ such that $\mathbf{e}_X^{val}(t_e)$ is defined,
       where $t_e = \mathbf{e}_X^{time}(t_1)$, and $x(t_1) \neq \mathbf{e}_X^{val}(t_e)$:
       If $X = Y$, $w \leftarrow w \cdot (1 - \exp(-q_{x(t_1)|\mathbf{u}_X(t_1)}(t_e - t_1)))$
       Else $w \leftarrow w \cdot \frac{1-\exp(-q_{x(t_1)|\mathbf{u}_X(t_1)}(t_e-t_1))}{1-\exp(-q_{x(t_1)|\mathbf{u}_X(t_1)}(t_e-t_2))}$
  3. return $w$
```

Figure 4: Importance sampling for CTBNs. Changes from Figure 2 are noted with asterisks.

The resulting algorithm is an instance of sequential importance sampling, and therefore suffers from its characteristic flaw: As the trajectory length increases, the distribution of the importance weights gets increasingly skewed, with most importance weights converging to zero exponentially quickly. Thus, the number of "relevant" samples gets increasingly small, and the estimates provided by the set of samples quickly become meaningless. A family of methods, commonly known as sequential Monte Carlo or particle filtering (Doucet et al., 2001), have been proposed in the setting of discrete-time processes to address this flaw.

```
Procedure CTBN-Particle-Filtering($\{X_0^i, w_0^i\}_{i=1...N}, t_{end}, \mathbf{e}$)
  1. $k \leftarrow 0, W_t \leftarrow 1, N_r \leftarrow N$
  2. For $i \leftarrow 1$ to $N$: $Pa_0^i \leftarrow i$, $w^i \leftarrow 1/N$
  Loop:
  3. For each i such that $t_k^i < t_{end}$ :
       $(X_{k+1}^i, t_{k+1}^i, w_{k+1}^i) \leftarrow$
           Sample-Segment($X_k^{Pa_k^i}, t_k^{Pa_k^i}, w^i, t_{end}, \mathbf{e}$)
       If $t_{k+1}^i \geq t_{end}$
           $N_{remain} \leftarrow N_r - 1$,
           $W_t \leftarrow W_t - w_{k+1}^i$
  4. $k \leftarrow k+1$
  5. If $N_r = 0$
       return $\{X_{m_i}^i, t_{m_i}^i, w_{m_i}^i, Pa_{m_i}^i\}_{i=1...N, m_i=1...n_i}$,
       where $n_i$ is the number of transitions of the $i^{th}$ particle
  6. Calculate $\widehat{N_{eff}}$ of all incomplete particles
  7. If $\widehat{N_{eff}} < N_{thr}$
       Sample $Pa_k^i$ according to $w_k^i$
       $w^i \leftarrow W_t \times 1/N_r$
    Else
       $w^i \leftarrow w_k^i$, $Pa_k^i \leftarrow Pa_{k-1}^i$
```

Figure 5: Particle Filtering for CTBNs

At a high level, these methods re-apportion our samples to focus more efforts on more relevant samples—those with higher weights.

The application of this idea to our setting introduces some subtleties because different samples are not generally synchronized. We could pick a time $t$ and run the algorithm in Figure 4 with $t_{end} = t$ so that samples are synchronized at $t$. We would re-apportion the weights and continue each trajectory from its state at $t$, first setting $Time(X)$ to be undefined for all $X$. However, choosing the proper synchronization time $t$ is a non-trivial problem which may depend on the evidence and the speed the system evolves.

Instead of synchronizing all the particles by the time, we can align particles by the number of transitions. If we let $t_i$ be the $i^{th}$ transition time and $X_i$ be the value of $X$ from $t_{i-1}$ to $t_i$, the following recursion holds.

$$\begin{aligned} P(X[0:t_n]) &= P(X_{1:n}, t_{1:n}, e_{[0:t_n)}) \\ &= P(X_{1:n-1}, t_{1:n-1}, e_{[0:t_{n-1})}) P(X_n | X_{n-1}) P(X_{[t_{n-1}, t_n)}, e_{[t_{n-1}, t_n)} | X_{n-1}, e_{t_{n-1}}). \end{aligned}$$

The weighted approximation of this probability is given by

$$P(X[0:t_n]) \approx \sum_{i=1}^{N} w(X^i[0:t_n]) \delta(X[0:t_n], X^i[0:t_n])$$

where $X^i[0:t_n)$ is the $i^{th}$ sample and $w(X^i[0:t_n))$ is the normalized weight of the $i^{th}$sample. According to Equation 8, the weight can be updated after every transition step. The weight update equation can be shown as

$$w(X^i[0:t_n)) \propto w(X^i[0:t_{n-1})) \frac{\tilde{L}_X(X^i[t_{n-1}:t_n))}{\tilde{L}'_X(X^i[t_{n-1}:t_n))}.$$

Thus, to sample multiple trajectories in parallel, we apply the CTBN importance sampling algorithm to each trajectory until a transition occurs. To avoid the degeneracy of the weights, we resample the particles when the estimated effective sample size $\widehat{N_{eff}} = \frac{1}{\sum_i (w_k^i)^2}$ is below a threshold $N_{thr}$. This procedure is similar to the regular particle filtering algorithm except that all particles are not synchronized by time but the number

```
Procedure CTBN-Particle-Smoothing($(\{X^i_{m_i}, t^i_{m_i}, w^i_{m_i}\}, t_{end}, \mathbf{e})$
            $i = 1 \ldots N, m_i = 1 \ldots M_i$
    1. $\sigma \leftarrow \emptyset$
    2. Choose $k$ with probability proportional to $w_i^{M_i}$
    3. set $Y = X_{M_k^k}$, $s \leftarrow M_k$, $t \leftarrow t_s^k$
    Loop:
    4. $\sigma_{[t_{s-1},s)} \leftarrow Y$
    5. If $\sigma$ is complete
          return $\sigma$
    6. For $j \leftarrow 1$ to $N$
          $w'_j \leftarrow$ Check-Weight$(Y, t, X^j_{s-1}, t^j_{s-1}, w^j_{s-1})$
    7. Choose $i$ with probability proportional to $w'_i$
    8. $Y \leftarrow X_i^s$, $t \leftarrow t_s^j$ $s \leftarrow s - 1$

    Procedure Check-Weight$(X, t, X_s, t_s, w_s)$
    1. If $t \leq t_s$ or $\mathbf{e}_{(ts,t)}$ contains a transition, or
          the value of $X$ and $X_s$ do not differ by only one variable
          return 0
    2. $\sigma_{[t_s,t)} \leftarrow X_s$, $\sigma(t) \leftarrow X$
    3. $w \leftarrow w_s \cdot \tilde{L}_X(\sigma_{[t_s,t_2]})$
    4. return $w$
```

Figure 6: Particle Smoothing for CTBNs

of transitions. To answer queries in the time interval $[0,T)$, we propagate the particles until all of their last transitions are greater than $T$.

Figure 5 shows the algorithm for generating $N$ trajectories from 0 to $T$ in a CTBN. It assumes that the initial values and the weights have already been sampled. The procedure *Sample-Segment* loops from line 3 to 7 in Figure 4 until a transition occurs, returns the transition time and variables value, and updates the corresponding weight for that segment. Note that we are approximating the distribution $P(X_{1:n}, t_{1:n}, e_{[0:t_n)})$ for all possible $n$. Therefore, we only propagate and re-apportion weights for particles that have not yet reached time $T$. Particles that have been sampled past $T$ are left untouched.

### 3.8 Particle Smoothing

Although the resampling step in the particle filtering algorithm reduces the skew of the weights, it leads to another problem: the diversity of the trajectories is also reduced since particles with higher weights are likely to be duplicated multiple times in the resampling step. Many trajectories share the same ancestor after the filtering procedure. A Monte Carlo smoothing algorithm using backward simulation addresses this problem (Godsill et al., 2004).

The smoothing algorithm for discrete-time systems generates trajectories using $N$ weighted particles $\{x_t^i, w_t^i\}$ from the particle filtering algorithm. It starts with the particles at time $T$, moves backward one step each iteration and samples a particle according to the product of its weight and the probability of it transitioning to the previously sampled particle. Specifically, in the first step, it samples $\widetilde{x}_T$ from particles $x_T^i$ at time $T$ with probability $w_T^i$. In the backward smoothing steps it samples $\widetilde{x}_t$ according to $w^i_{t|t+1} = w^i_t f(\widetilde{x}_{t+1}|x^i_t)$, where $f(\widetilde{x}_{t+1}|x^i_t)$ is the probability that the particle transitions from state $x^i_t$ to $\widetilde{x}_{t+1}$. The resulting trajectory set is an approximation of $P(x_{1:T}|y_{1:T})$ where $y_{1:T}$ is the observation.

This idea can be used in our setting with modification. Given the filtered particles $\{X^i_{m_i}, t^i_{m_i}, w^i_{m_i}\}$,we need to sample both variable values and transition times at each step when we move backward. There are two main differences from the algorithm in Godsill et al. (2004): There are fewer than $N$ particles that can be

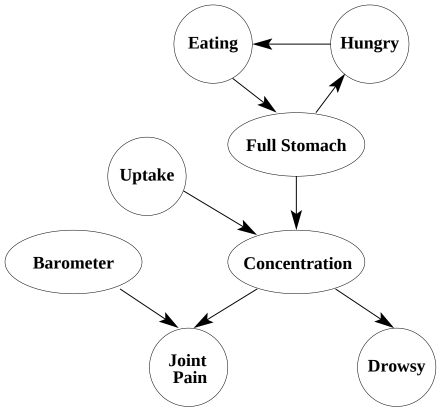


Figure 7: Drug Effect Network

used at the beginning steps of the backward smoothing since the trajectories do not have exactly the same number of transitions, and not all particles at step $n$ can be considered as candidates to move backward. A particle $\{X_n^i, t_n^i, w_n^i\}$ is a valid candidate as the predecessor for $\{\widetilde{X}_{n+1}, \widetilde{t}_{n+1}\}$ only if (1) $t_n^i < \widetilde{t}_{n+1}$, (2) the values of $X_n^i$ and $\widetilde{X}_{n+1}$ differ in only one variable (thus a single transition is possible), and (3) $\mathbf{e}_{(t_n^i, \widetilde{t}_{n+1})}$ contains no transitions.

Figure 6 shows the smoothing algorithm which generates a trajectory from the filtering particles. We apply the algorithm $N$ times to sample $N$ trajectories. These equally weighted trajectories can be used to approximate the smoothing distribution $P(X_{[0,T)}|\mathbf{e})$. Generating one trajectory with this smoothing process requires considering all the particles at each step. The running time of sampling $N$ trajectories using particle smoothing is $N$ times of that of particle filtering.

## 4. Experimental Results

In this section, we report on the performance of our algorithm on synthetic networks and a network built from a real data set of people's life histories. We tested our algorithm's accuracy for the task of inference and parameter estimation. We also compare our algorithms with other approximate inference algorithms for CTBNs: the method based on the expectation propagation in Saria et al. (2007) and the method based on Gibbs sampling in El-Hay et al. (2008).

All the algorithms we used in the experiments were implemented in the same code base to make fair comparisons. We tried our best to optimize all the code. The implementations are general so that they can be applied to any CTBN model. Our implementation of EP is adapted from that of Saria et al. (2007) who were kind enough to share their code. The code base is described in Shelton et al. (2010) and is available from the authors' website.

### 4.1 Networks

In our experiments, different types of network structures were used, including the drug effect network (Nodelman et al., 2002), a chain-structured network, and the BHPS network (Nodelman et al., 2005b). All the networks are at the upper size limit for the exact inference algorithm so that we can compare our result to the true value.

**Drug Effect Network:** The drug effect network is a toy model of the effect of a pain-relief medicine. It has 8 (5 binary and 3 ternary) variables. The structure of the network is shown in Figure 7. At $t = 0$ the person is not hungry, is not eating, has an empty stomach and is not drowsy. He has joint pain due to the falling barometric pressure and takes the drug to alleviate the pain.

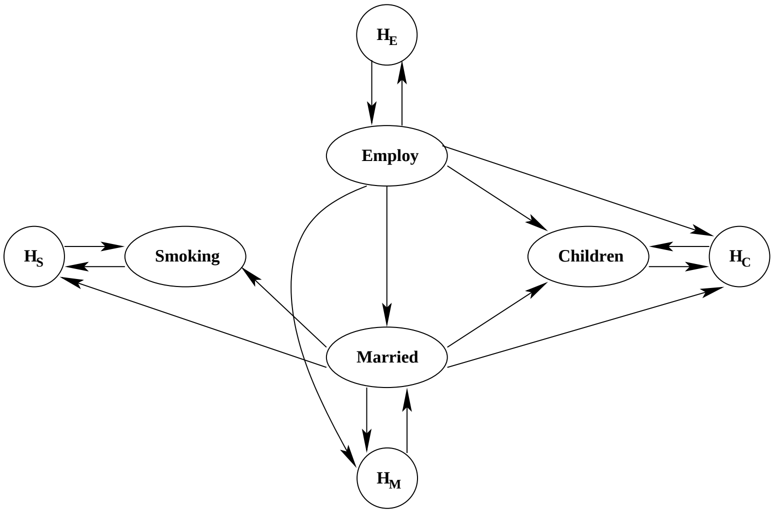


Figure 8: British Household Panel Survey Network

**Chain Structured Network:** The chain network contains five nodes $X_0, \ldots, X_5$, where $X_i$ is the parent of $X_{i+1}$ for $i < 5$. Each node has five states, $s_0, \ldots, s_4$. $X_0$ (usually) cycles in two loops: $s_0 \to s_1 \to s_3 \to s_0$ and $s_0 \to s_2 \to s_4 \to s_0$. All the other nodes stay at their current state if it matches their parent and otherwise transition to their parent's state with a high probability. Each variable starts in state $s_0$.

More specifically, the intensity matrix of $X_0$ is

$$\mathbf{Q}_{X_0} = \begin{bmatrix} -2.02 & 1 & 1 & 0.01 & 0.01 \\ 0.01 & -2.03 & 0.01 & 2 & 0.01 \\ 0.01 & 0.01 & -2.03 & 0.01 & 2 \\ 2 & 0.01 & 0.01 & -2.03 & 0.01 \\ 2 & 0.01 & 0.01 & 0.01 & -2.03 \end{bmatrix}.$$

For all other nodes, the off-diagonal elements of the intensity matrices are given by

$$q_{s_i,s_j|\mathbf{u}=s_k} = \begin{cases} 0.1 & \text{if } i \neq j \text{ and } j \neq k, \\ 10 & \text{if } i \neq j \text{ and } j = k. \end{cases}$$

**BHPS Network:** This network was learned from the British Household Panel Survey (BHPS) (ESRC Research Centre on Micro-social Change, 2003) data set. The data set provides information about British citizens. The data are collected yearly by asking thousands of households questions such as household organization, employment, income, wealth and health. Similar to Nodelman et al. (2005b), we keep a small set of variables so that exact inference could be applied. We chose four variables: employ (ternary: student, employed, unemployed), children (ternary: 0, 1, 2+), married (binary: not married, married), and smoking (binary: non-smoker, smoker), and we assumed there is a hidden variable (binary) for each of those four variables. We trained the network on 8935 trajectories of people's life histories. We applied the structural EM algorithm in Nodelman et al. (2005b) and learned the structure of the network shown in Figure 8. We then estimated the parameters of the network using the EM algorithm and exact inference. We consider the learned model as the true BHPS network model for these experiments.

### 4.2 Evaluation Method

We evaluated the performance of the approximate inference algorithms in two tasks: the inference task of answering queries given evidence and the learning task of parametric learning with partially observed data.

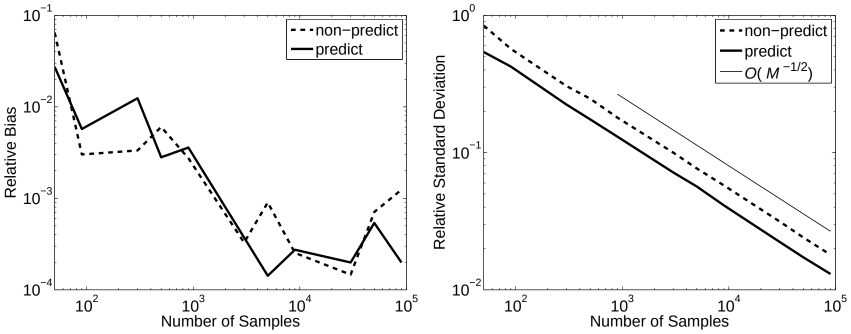


Figure 9: Relative bias and standard deviation of sampling with and without predictive lookahead.

In the inference task, each evidence is a partially observed trajectory of the CTBN network. The evidence is generated using two methods. The first method is to set it manually. The second is to generate a trajectory using the forward sampling algorithm and randomly remove some parts of the sampled trajectory. In particular, we repeated the following procedure $n$ times: for each variable, we randomly removed the information of the trajectory from $t_s$ to $t_s+\gamma T$, where $T$ is the total length of the trajectory, $t_s$ is randomly sampled from the $[0,T-\gamma T]$ uniform distribution and $\gamma<1$. After we run the removing procedure $n$ times, there are at most $n\gamma$ time units of information missing for each variable. In all comparisons, this procedure was applied once and the same evidence was given to all algorithms.

In our experiments, we set our query to be one of three types: the expected total amount of time a variable $X$ stays on some state $x_i$, the expected total number of times that a variable transitions from state $x_i$ to state $x_j$, or the distribution of variable at time $t$.

For each query, we ran the sampling based algorithms with different sample sizes, $M$. For each sample size, we ran the experiment $N$ times. We calculated our query according to Equation 6 and compared the result to the true value calculated using exact inference. We used two metrics: the relative bias $\frac{|\sum v_M - v^*|}{v^* N}$, where $v_M$ is the query value of sampling algorithm with sample size $M$, and $v^*$ is the true value; and the relative standard deviation $\frac{\sigma_M}{v^*}$ where $\sigma_M$ is the standard deviation from the true value when sample size is $M$. For each sample size, we also recorded the average running time $\overline{t_M}$ of each experiment and used $\overline{t_M}$ to evaluate the efficiency of the algorithm.

In the learning task, we used the sampling algorithms to estimate the parameters of a CTBN network given some partially observed data. Monte Carlo EM (Wei and Tanner, 1990) was applied in this task: In each iteration, we used the sampling based algorithm to estimate the expected sufficient statistics given the incomplete data and used Equation 4 to compute the parameters.

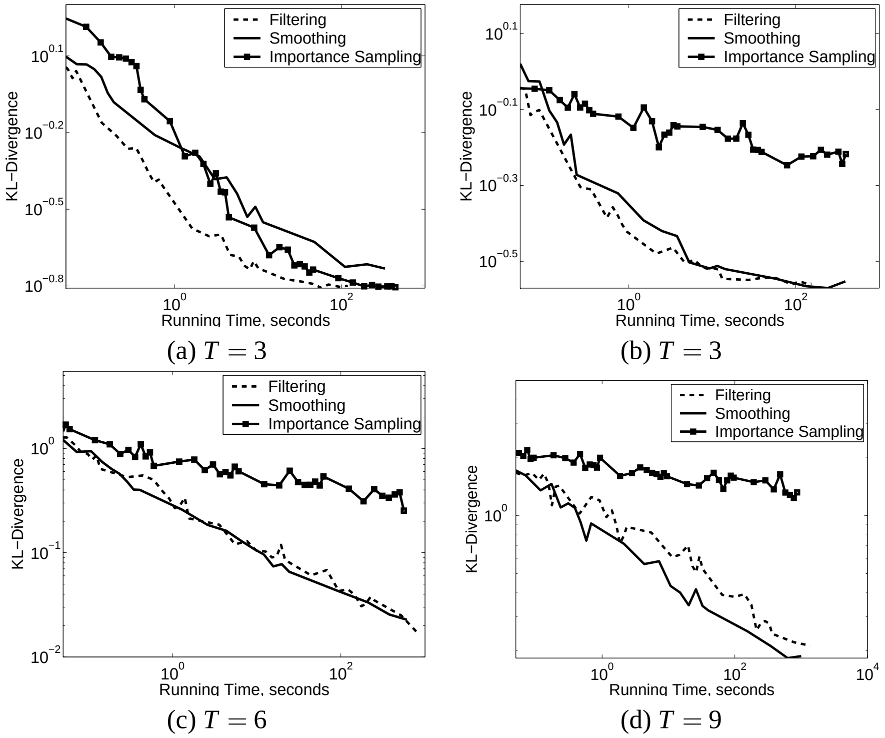
The training data were generated by sampling trajectories from the true model and randomly removing some portion of the information as described above. We sampled another set of trajectories from the true model as the testing data. We calculated the log-likelihood of the testing data under the learned model to evaluate the learning accuracy.

### 4.3 Inference Experimental Results

In this section, we evaluate the performance of our importance sampling based algorithms in answering queries and compare with the EP algorithm in Saria et al. (2007) and the Gibbs sampling algorithm in El-Hay et al. (2008).

#### 4.3.1 Comparison of Importance Sampling and Predictive Lookahead

We first tested the importance sampling algorithm and the predictive lookahead modification using the drug effect network. We set the observed evidence: on $t=[0,1)$ the stomach is empty, on $t=[0.5,1.2)$ the



(a) $T = 3$ (b) $T = 3$

(c) $T = 6$ (d) $T = 9$

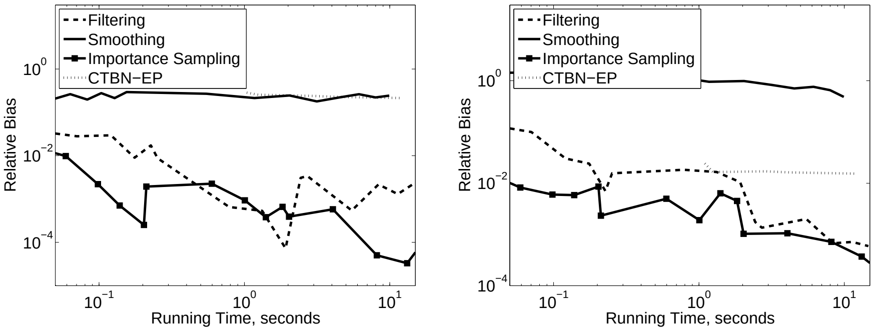
Figure 10: Time-efficiency comparison of particle filtering, smoothing and importance sampling

barometer is falling, and on $t = [1.5, 2.5)$ he is drowsy. Our query is the expected total amount of time that he has no joint pain on $[0, 2.5)$. (The true value is 0.1093). We ran the two algorithms with sample sizes, $M$, from 5 to 90000. For each sample size, we ran the algorithms $N = 1000$ times. The results are shown in Figure 9. Both algorithms achieve the correct result when the sample size is large. The standard deviation decreases at a rate of $O(\frac{1}{\sqrt{M}})$ (shown by the thin solid line). The sampling algorithm with prediction achieves lower standard deviation than the non-prediction version.

### 4.3.2 IMPORTANCE SAMPLING, PARTICLE FILTERING AND SMOOTHING

We then used the chain network to evaluate the efficiency of the importance sampling, particle filtering, and smoothing algorithms. We assumed that only $X_4$ was observed in this experiment. We used four different evidences. The first one is a simple evidence: only part of the behavior of $X_4$ is observed: on $[1, 1.7)$, $X_4 = s_3$, and on $[2, 2.5)$, $X_4 = s_2$. For the other three, the behavior of $X_4$ is fully observed during the interval $[0, T)$, where $T = 3, 6, 9$. This is done by forward sampling a trajectory from 0 to $T$ and keeping only the information about $X_4$. Our query is the marginal distribution $P(X_2(\frac{T}{2})|\mathbf{e}_{[0,T)})$. Note that this is the most difficult case for the importance sampling algorithm since the chain network is nearly deterministic. We recorded the average running time and KL-divergence between the estimated and true distributions, for each sample size across $N = 300$ trials.

Figure 10 shows the efficiency of the three algorithms. In Figure 10(a), we used the simple evidence. In Figure 10 (b)-(d), we used the evidence with $X_4$ fully observed and $T = 3, 6, 9$ respectively. In all four cases, the particle filtering and smoothing algorithms both outperform the importance sampling algorithm when the sample size is small (small running time). For simple evidence (Figure 10(a)), the importance sampling algorithm achieves comparable performance when the sample size is large. When the evidence is complicated (Figure 10 (b)-(d)), the error of importance sampling is large, even when we use very large



(a) Concentration

(b) Joint pain

Figure 11: Comparison to expectation propagation: Drug Network

sample sizes. When the trajectory is short, the particle filtering algorithm is slightly better than the particle smoothing algorithm. This is because the filtering algorithm can generate more samples than the smoothing algorithm with the same running time. However, as the trajectory length increases, the particle smoothing algorithm outperforms the filtering algorithm due to particle diversity problems.

### 4.3.3 COMPARISON OF IMPORTANCE SAMPLING AND EP

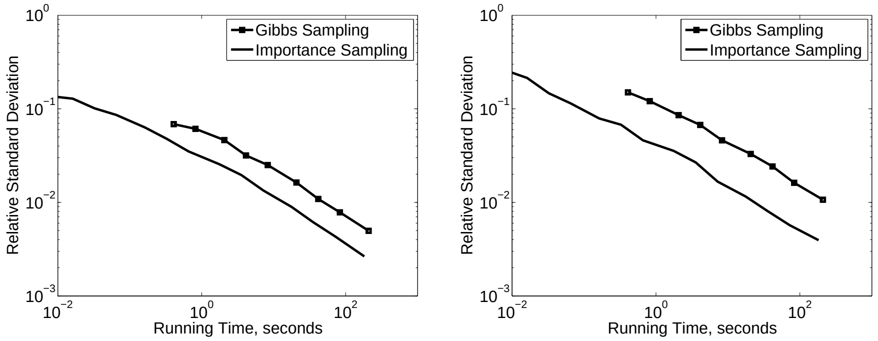
We also compared our three sampling algorithms to the approximate inference algorithm based on expectation propagation in Saria et al. (2007). We did not use their adaptive splitting method (for reasons we explain below). Even without the adaptive splitting, their method still differs from that of Nodelman et al. (2005a), in that it allows asynchronous propagation of messages along time.

We used the same evidence as in Section 4.3.1 on the drug effect network and answered two queries: the total amount of time that the concentration is low and the total amount of time the person has no joint pain. For the EP algorithm, we first tried a segmentation that split the timeline at the evidence. We then gradually decreased the time interval of the segments to 0.15. The results of accuracy with respect to running time are shown in Figure 11. The importance sampling algorithm and the particle filtering algorithm outperforms the EP algorithm in answering both queries. Among the sampling-based algorithms, the importance sampling algorithm performs the best and the smoothing algorithm is the worst. This is not surprising given that most of the nodes are binary. At each transition time, the sampled trajectory has no choice as to the next state. Therefore, smoothing (or filtering) has less effect as there is no need to intelligently select the next state. However, the extra computation time for resampling and backward simulation makes the filtering and smoothing algorithm less efficient.

As mentioned above, we did not employ the adaptive splitting method of Saria et al. (2007). It would not have changed our results much. The left-most points in our Figure 11 correspond to the minimum number of splits. (They are as fast as possible.) The right-most points of the Figure 11 correspond to many fine splits, and are about as accurate as possible, and we can see that the accuracy has flattened out. So, while the horizontal widths of the EP curves would have been shortened (by allowing for the better accuracy in less time), the vertical spread would have been approximately the same. In neither plot of Figure 11 would this have made a large difference in the comparisons to our sampling method.

### 4.3.4 COMPARISON OF IMPORTANCE SAMPLING AND GIBBS SAMPLING

We compared our importance sampling algorithm to the Gibbs Sampling algorithm in El-Hay et al. (2008). We used three CTBN network models: the drug effect network, the BHPS network and the chain structured network. For each network, we randomly generated evidence using the procedure described in Section 4.2.



(a) Query of Time

(b) Query of Transition

Figure 12: Comparison to Gibbs Sampling: Drug Network. Note burn-in time for Gibbs Sampling is not included (3.94 seconds on average).

We ran the procedure 4 times for each variable and each time, we removed 20% of the content. Thus, there are at most 80% information missing for each variable.

For the importance sampling algorithm, we chose the sample size $M$ from 10 to 500000. For Gibbs sampling algorithm, we chose the sample size $M$ from 10 to 5000. We ran the experiments for each sample size $N = 100$ times and recorded the average running time for each algorithm. For Gibbs sampling algorithm, we first ran 100 “burn-in” iterations for each sample size before we sample trajectories from the sampler. The time spent on the “burn-in” iterations was not included in the final running time.

For the drug effect network, the evidence trajectory begins at time $t = 0$ and ends at time $t = 5$. We asked two queries: the expected total amount of time the person’s stomach is half full, and the expected number of times that the person’s stomach changes from empty to half full.

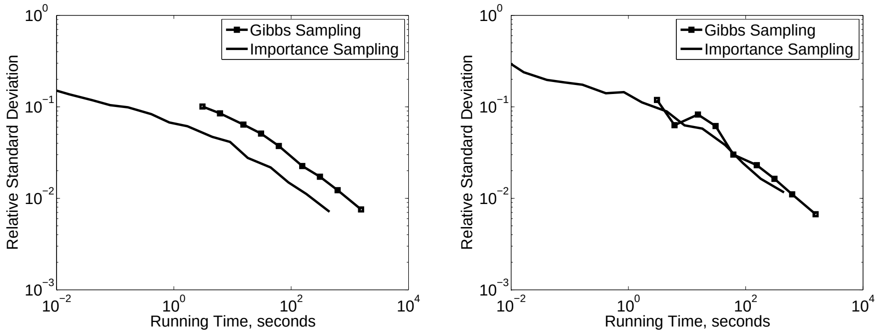
Using enough running time (sample size), we observed that both algorithms could answer the queries accurately (with a relative bias below 0.1%). The decreasing of the relative standard deviation with respect to the running time of the two algorithms are shown in Figure 12. The average “burn-in” time for Gibbs sampler is about 3.94 seconds. From the figure, we can see that importance sampling outperforms Gibbs sampling in answering both queries.

For the BHPS network, we set the evidence from $t = 0$ to $t = 50$ (years). We asked similar queries: the expected total amount of time a person’s employment status is as a student and the expected number of times that he becomes employed. We chose the same sample sizes as on the drug effect network and ran each sample size $N = 100$ times. Figure 13 shows the result of the decreasing of the standard deviation of the two algorithms. The average “burn-in” time for Gibbs sampling algorithm in this experiment is 30.88 seconds.

We achieved similar result as the experiments with the drug effect network. The importance sampling algorithm outperformed the Gibbs sampling algorithm in answering the query of time. The performances on the query of transitions are almost the same.

In both networks, importance sampling outperformed Gibbs sampling in three of the four cases, even when the running time on “burn-in” iterations was not considered. To achieve the same accuracy and standard deviation, Gibbs sampling algorithm requires fewer samples. This is because for each variable, Gibbs sampling samples from the true posterior distribution given the evidence and its Markov blanket. However, sampling from the true posterior distribution is computational costly, since it requires repeatedly computing the conditional cumulative distribution function. Using the same amount of time, importance sampling can sample far more trajectories, which outperforms Gibbs sampling.

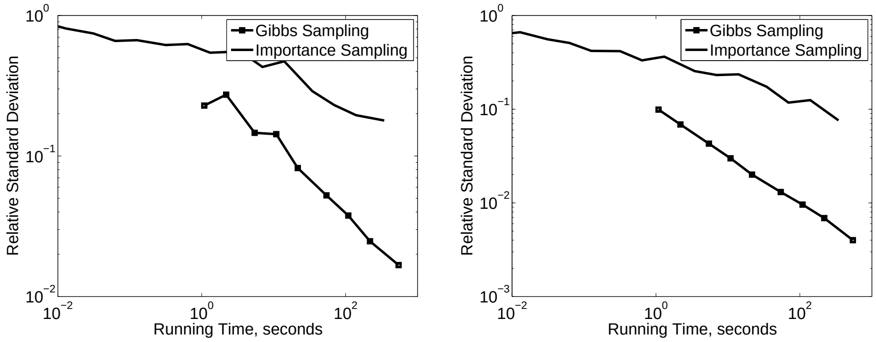
We last compared these two algorithms using the chain structured network. The evidence trajectory begins at time $t = 0$ and ends at time $t = 5$. We set the queries to be the expected total amount of time $X_2$ stays in



(a) Query of Time

(b) Query of Transition

Figure 13: Comparison to Gibbs Sampling: BHPS Network. Note burn-in time for Gibbs Sampling is not included (30.88 seconds on average).



(a) Query of Time

(b) Query of Transition

Figure 14: Comparison to Gibbs Sampling: Chain Network. Note burn-in time for Gibbs Sampling is not included (11.42 seconds on average).

state $s_1$ and the expected number of times that $X_2$ transitions from $s_0$ to $s_1$. Figure 14 shows the result over $N = 100$ runs. The average "burn-in" time for Gibbs sampling algorithm in this experiment is 11.42 seconds.

The Gibbs sampling algorithm achieved better performance in this experiment. The result is not surprising. As we have mentioned before, the chain structured network is nearly deterministic, and it is the hardest case for the importance sampling algorithm. We further examined the randomly generated evidence. The only observed state on $X_0$ is $s_0$, which makes this experiment even harder for the importance sampling algorithm. However, it is a very easy case for the Gibbs sampling algorithm since it is nearly deterministic and is structurally simple. (There are only at most one parent and one child for each node.) Although importance sampling can generate many more samples in the same period of time, most of these samples are trajectories with very small weights.

### 4.4 Parameter Estimation Experimental Results

In this section, we evaluate the performance of importance sampling algorithm on parameter estimation and compare to the Gibbs sampling algorithm and the EP algorithm.

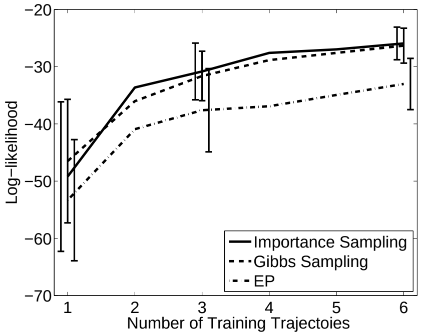


Figure 15: Learning results for the drug effect network. Standard deviations across training data selection (and random samples for the sampling methods) are shown jittered slightly for clarity.

We used the drug effect network for this experiment. We sampled increasing We sampled increasing numbers of trajectories of 5 time lengths. To hide part of the trajectory, we did the following: In each iteration, for each variable we randomly selected a time window of 0.5 time lengths and removed the content in that window. We repeated this until we dropped 50% of the content of the trajectory. We used these incomplete trajectories as our training data. We sampled another 50 trajectories with the same length as our testing data.

To estimate the parameters of the CTBN network, we followed the EM algorithm in Nodelman et al. (2005b). When calculating the expected sufficient statistics, importance sampling, Gibbs sampling, and expectation propagation were used. Therefore, the likelihood in the E-step was calculated approximately. In our experiment, we fixed the total number of iterations for the EM algorithm at 15. In each iteration, we compared the calculated likelihood to the likelihood in the previous iteration. If the likelihood decreased, we kept the parameters in the previous iteration.

We chose the initial parameters for the EM algorithm by sampling the diagonal elements of the conditional intensity matrices from the Gamma distribution with parameters $(0.5, 1)$ and sampling the transition probabilities from a Dirichlet distribution. We randomly sampled 5 models as the initial parameters for the EM algorithm. For each initial parameter set, we ran the EM algorithm 10 times for the sampling methods (and once for EP which is deterministic). We evaluated the learning accuracy by calculating the average log-likelihood of the testing data on the 50 learned networks. To compare the running efficiency of the two sampling-based algorithms, we fixed the total amount of time for the sampler to generate samples in each EM iteration to be the same time as the EP algorithm took (approximately 23 seconds). For the Gibbs sampling algorithm, we dropped the first 50 trajectories as “burn-in” iterations. Figure 15 shows the results as we increased the number of training trajectories from 1 to 6.

All algorithms obtain higher log-likelihood on the testing data when we increase the number of training trajectories. The sampling methods do better (and have lower variation), especially as the data size grows.

## 5. Conclusion

We have presented an approximate inference algorithm with two variations based on importance sampling. We naturally extended the algorithm to sequential Monte Carlo methods such as particle filtering and smoothing in CTBNs. We applied our sampling algorithm to synthetic networks and a network from real data. We evaluated the efficiency of our algorithms and compared to other approximate inference algorithms based on expectation propagation and Gibbs sampling. Our importance sampling algorithm outperformed both in most of the experiments presented in this paper. In the situation of a highly deterministic system, Gibbs sampling performed better.

The networks used in this paper are at the upper size limit for exact computation. For example, calculating the expected sufficient statistics of the chain structured network given evidence takes more than two days using exact inference. Thus, approximate inference methods are critical for tracking, predicting, and learning in continuous time Bayesian networks for real applications. Our importance sampling based algorithms are fast, simple to implement and can be used to calculate the expected value of any function of a trajectory, including the expected sufficient statistics necessary for expectation-maximization for parameter estimation with missing data.

## Acknowledgments

We would like to thank Suchi Saria for sharing her EP code and Tal El-Hay for sharing his Gibbs sampling code. This work was funded by AFOSR (FA9550-07-1-0076) and DARPA (HR0011-09-1-0030).

## References

Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor S. Trivedi. *Queueing Networks and Markov Chains*. John Wiley & Sons, Inc., 1998.

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

Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, 39:1–38, 1977.

Arnaud Doucet, Nando de Freitas, and Neil Gordon, editors. *Sequential Monte Carlo Methods in Practice.* Springer-Verlag Telos, 2001.

Tal El-Hay, Nir Friedman, and Raz Kupferman. Gibbs sampling in factorized continuous-time Markov processes. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, 2008.

ESRC Research Centre on Micro-social Change. British household panel survey. Computer Data File and Associated Documentation. http://www.iser.essex.ac.uk/bhps, 2003. Colchester: The Data Archive.

Yu Fan and Christian R. Shelton. Sampling for approximate inference in continuous time Bayesian networks. In *Proceedings of Tenth International Symposium on Artificial Intelligence and Mathematics*, 2008.

Robert M. Fung and Kuo-Chu Chang. Weighing and integrating evidence for stochastic simulation in Bayesian networks. In *Proceedings of the Fifth Annual Conference on Uncertainty in Artificial Intelligence*, pages 209–220, 1989.

Simon Godsill, Arnaud Doucet, and Mike West. Monte Carlo smoothing for non-linear time series. *Journal of the American Statistical Association*, 99:156–168, 2004.

Ralf Herbrich, Thore Graepel, and Brendan Murphy. Structure from failure. In *Proceedings of the 2nd USENIX workshop on Tackling computer systems problems with machine learning techniques*, pages 1–6, 2007.

Tim Hesterberg. Weighted average importance sampling and defensive mixture distributions. *Technometrics*, 37(2):185–194, 1995.

Kin Fai Kan and Christian R. Shelton. Solving structured continuous-time Markov decision processes. In *Proceedings of Tenth International Symposium on Artificial Intelligence and Mathematics*, 2008.

Thomas P. Minka. Expectation propagation for approximate Bayesian inference. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, pages 362–369, 2001.

Brenda Ng, Avi Pfeffer, and Richard Dearden. Continuous time particle filtering. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 1360–1365, 2005.

Uri Nodelman and Eric Horvitz. Continuous time Bayesian networks for inferring users' presence and activities with extensions for modeling and evaluation. Technical Report MSR-TR-2003-97, Microsoft Research, December 2003.

Uri Nodelman, Christian R. Shelton, and Daphne Koller. Continuous time Bayesian networks. In *Proceedings of the Eighteenth International Conference on Uncertainty in Artificial Intelligence*, pages 378–387, 2002.

Uri Nodelman, Christian R. Shelton, and Daphne Koller. Learning continuous time Bayesian networks. In *Proceedings of the Nineteenth International Conference on Uncertainty in Artificial Intelligence*, pages 451–458, 2003.

Uri Nodelman, Daphne Koller, and Christian R. Shelton. Expectation propagation for continuous time Bayesian networks. In *Proceedings of the Twenty-First International Conference on Uncertainty in Artificial Intelligence*, pages 431–440, 2005a.

Uri Nodelman, Christian R. Shelton, and Daphne Koller. Expectation maximization and complex duration distributions for continuous time Bayesian networks. In *Proceedings of the Twenty-First International Conference on Uncertainty in Artificial Intelligence*, pages 421–430, 2005b.

James R. Norris. *Markov Chains*. Cambridge University Press, 1997.

Carl Adam Petri. *Kommunikation mit Automaten*. PhD thesis, University of Bonn, 1962.

Suchi Saria, Uri Nodelman, and Daphne Koller. Reasoning at the right time granularity. In *Proceedings of the Twenty-third Conference on Uncertainty in AI*, pages 421–430, 2007.

Ross D. Shachter and Mark A. Peot. Simulation approaches to general probabilistic inference on belief networks. In *Proceedings of the Fifth International Conference on Uncertainty in Artificial Intelligence*, pages 221–234, 1989.

Christian R. Shelton, Yu Fan, William Lam, Joon Lee, and Jing Xu. Continuous time Bayesian network reasoning and learning engine. *Journal of Machine Learning Research*, 11(Mar):1137–1140, 2010.

Charles A. Sutton and Michael I. Jordan. Probabilistic inference in queueing networks. In Armando Fox and Sumit Basu, editors, *Proceedings of Third Workshop on Tackling Computer Systems Problems with Machine Learning Techniques*, 2008.

Greg C. G. Wei and Martin A. Tanner. A Monte Carlo implementation of the EM algorithm and the poor man's data augmentation algorithms. *Journal of the American Statistical Association*, 85(411):699–704, 1990.

Jing Xu and Christian R. Shelton. Continuous time Bayesian networks for host level network intrusion detection. In *European Conference on Machine Learning*, pages 613–627, 2008.