Journal of Artificial Intelligence Research 51 (2014) 725–778 Submitted 04/14; published 12/14

Tutorial on Structured Continuous-Time Markov Processes

Christian R. Shelton CSHELTON@CS.UCR.EDU
University of California, Riverside

Gianfranco Ciardo CIARDO@IASTATE.EDU
Iowa State University

Abstract

A continuous-time Markov process (CTMP) is a collection of variables indexed by a continuous quantity, time. It obeys the Markov property that the distribution over a future variable is independent of past variables given the state at the present time. We introduce continuous-time Markov process representations and algorithms for filtering, smoothing, expected sufficient statistics calculations, and model estimation, assuming no prior knowledge of continuous-time processes but some basic knowledge of probability and statistics. We begin by describing "flat" or unstructured Markov processes and then move to structured Markov processes (those arising from state spaces consisting of assignments to variables) including Kronecker, decision-diagram, and continuous-time Bayesian network representations. We provide the first connection between decision-diagrams and continuous-time Bayesian networks.

1. Tutorial Goals

This tutorial is intended for readers interested in learning about continuous-time Markov processes, and in particular compact or structured representations of them. It is assumed that the reader is familiar with general probability and statistics and has some knowledge of discrete-time Markov chains and perhaps hidden Markov model algorithms.

While this tutorial deals only with Markovian systems, we do not require that all variables be observed. Thus, hidden variables can be used to model long-range interactions among observations. In these models, at any given instant the assignment to *all* state variables is sufficient to describe the future evolution of the system. The variables themselves real-valued (continuous) times. We consider evidence or observations that can be regularly spaced, irregularly spaced, or continuous over intervals. These evidence patterns can change by model variable and time.

We deal exclusively with discrete-state continuous-time systems. Real-valued variables are important in many situations, but to keep the scope manageable, we will not treat them here. We refer to the work of Särkkä (2006) for a machine-learning-oriented treatment of filtering and smoothing in such models. The literature on parameter estimation is more scattered. We will further constrain our discussion to systems with finite states, although many of the concepts can be extended to countably infinite state systems.

We will be concerned with two main problems: inference and learning (parameter estimation). These were chosen as those most familiar to and applicable for researchers in artificial intelligence. At points we will also discuss the computation of steady-state properties, especially for model for which most research concentrates on this computation.

©2014 AI Access Foundation. All rights reserved.

The first section (Section 2) covers the basics of flat (unstructured state-space) continuous-time Markov processes. The remaining sections discuss compact representations. This tutorial's goal is to make the mathematical foundations clear and lay out the current research landscape so that more detailed papers can be read more easily.

1.1 Related Models

There are many non-Markov continuous time models. Gaussian processes (Williams, 1998) are the best-known and model continuous-valued processes. For discrete-valued processes, most models build upon Poisson processes, or more general marked processes. As a Poisson process is memoryless, to make an interesting model, researchers usually generalize to allow the rate of an event to be a function of the process's history.

Poisson networks (Rajaram, Graepel, & Herbrich, 2005) constrain this function to depend only on the counts of the number of events (possibly of different event types) during a finite time window. The cascade of Poisson process model (Rajaram et al., 2005) defines the rate function to be the sum of a kernel applied to each historic event. The kernel has parameters for the effect of time passing, overall event rate, and chance that one type of event follows another. Piecewise-constant intensity models (PCIMs) (Gunawardana, Meek, & Xu, 2012; Parikh, Gunamwardana, & Meek, 2012) define the intensity function as a decision tree, with internal nodes' tests drawn from a set of pre-specified binary tests of the history. Forest-based point processes (Weiss & Page, 2013) extend this by allowing the intensity function to be the product of a set of functions, each a PCIM-like tree. Didelez (2008) presents a generalization of the continuous-time Bayesian networks (see Section 5) to inhomogeneous point processes, but without specific parameterizations or algorithms.

1.2 Why Continuous Time

Contemporary computers are discrete-time computation engines (or at least present a model of one). Therefore, why would we consider a continuous-time model? The quickest answer is by analogy: We build models of non-temporal systems employing real-valued variables. The tools of linear algebra, calculus, and the like allow us to derive and analyze these algorithms and methods. Yet, in the end they will be implemented on discrete-valued computers with finite memory and precision. However, we find the abstraction of continuous-valued variables useful and only make approximations during the final implementation when employing fixed- or floating-point precision arithmetic.

Similarly, it is productive to treat time as a continuous quantity. It allows us to more naturally discuss and reason about systems in which

1. Events, measurements, or durations are irregularly spaced,
2. Rates vary by orders of magnitude, or
3. Durations of continuous measurement need to be expressed explicitly.

All of these happen in asynchronous systems. Most dynamic systems of interest are asynchronous: events or measurements (or both) do not occur based on some global clock. Social networks, phylogenetic trees, and computer system logs are just some examples.

Note that while the underlying system model is continuous-time, observations and measurements of that model need not be continuous. We directly treat discrete-time observations, both at regular and irregular intervals.

1.3 Why Not Discrete Time

Clearly for any given continuous-time system specification, some discretization of time values could be made without introducing too much approximation error. Such a conversion of time from real-valued to integral makes it mathematically more difficult to be flexible about how to treat time algorithmically. This makes the development of computationally efficient algorithms more difficult. For instance, in a discrete-time model, it is natural to have computations proceed one time "step" at a time. However, for uneventful times, this can be computationally overly burdensome. With a continuous-time model, because there is no natural time step, it is simpler to think about methods that can "jump" over such uneventful time periods. Additionally, there are a few oddities about Markov chains built by discretizing continuous time. Finally, the full system specification may not be known when the discretization must be selected (for instance, if parameters must be estimated).

1.3.1 Time Discretization and Markovian-ness

Consider the two-state Markov chain X described by the stochastic matrix[1]

$$\boldsymbol{T_1} = \begin{bmatrix} 0.75 & 0.25 \\ 0.5 & 0.5 \end{bmatrix} . \tag{1}$$

The elements of $\boldsymbol{T_1}$ are the probabilities $p(X_t \mid X_{t-1})$ for each value of X_t and X_{t-1}. Over one time unit, the probability of moving from state 1 to state 2 is 0.25, for example.

If $\boldsymbol{T_1}$ describes a continuous-time system, sampled at a period of 1 time unit, there should be a matrix $\boldsymbol{T_{1/2}}$ describing the same system, sampled at a period of $\frac{1}{2}$ time unit (or twice the sampling *rate*). Indeed there is:

$$\boldsymbol{T_{1/2}} = \begin{bmatrix} 0.83 & 0.17 \\ 0.33 & 0.67 \end{bmatrix} . \tag{2}$$

This can be verified:

$$\begin{aligned} P(X_t = j \mid X_{t-1} = i) &= \sum_k P(X_{t-1/2} = k \mid X_{t-1} = i) P(X_t = j \mid X_{t-1/2} = k) \\ \boldsymbol{T_1}(i,j) &= \sum_k \boldsymbol{T_{1/2}}(i,k) \boldsymbol{T_{1/2}}(k,j) \\ \boldsymbol{T_1} &= \boldsymbol{T_{1/2}} \boldsymbol{T_{1/2}} . \end{aligned}$$

That is, $\boldsymbol{T_{1/2}}$ is the matrix square root of $\boldsymbol{T_1}$.

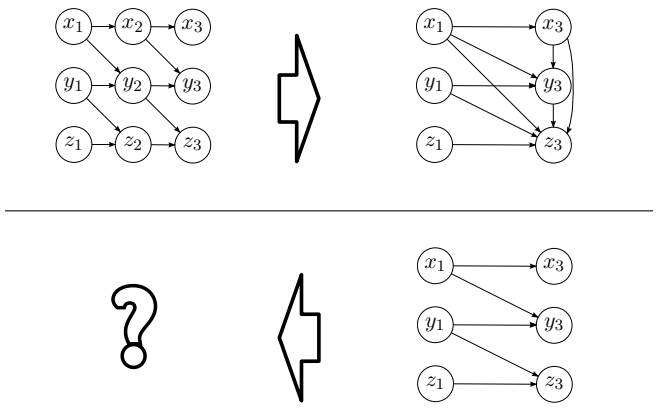
Now take a different two-state Markov chain transition matrix

$$\boldsymbol{S_1} = \begin{bmatrix} 0.1 & 0.9 \\ 0.9 & 0.1 \end{bmatrix} \tag{3}$$

and construct the corresponding transition matrix at half the sampling period, $\boldsymbol{S_{1/2}}$:

$$\boldsymbol{S_{1/2}} = \begin{bmatrix} 0.5 + .447i & 0.5 - .447i \\ 0.5 + .447i & 0.5 - .447i \end{bmatrix} . \tag{4}$$

1. We will use *row*-stochastic matrices exclusively in this tutorial. While column-stochastic matrices are often used for discrete time, row-stochastic matrices are more common in continuous time.



(a) unrolled DBN (b) marginalized DBN

Figure 1: Example of (a) a DBN unrolled, and (b) the same DBN marginalized to twice the sampling periodicity

There is no real-valued stochastic matrix describing the same processes as $\boldsymbol{S_1}$, but at half the sampling periodicity. Put differently, there is no two-state continuous-time Markov system that when sampled at a rate of 1 time unit produces the Markov chain with transition matrix $\boldsymbol{S_1}$.

The problem in generating $\boldsymbol{S_{1/2}}$ arises because $\boldsymbol{S_1}$ has a negative eigenvalue (by contrast, all eigenvalues of $\boldsymbol{T_1}$ are positive). In general, only stochastic matrices with all positive eigenvalues correspond to a continuous-time Markov process sampled at a given periodicity. This can be viewed in two ways. First, it means that the set of continuous-time Markov processes is smaller than the set of discrete-time Markov processes. Second, it means that there are processes that are Markovian only when sampled at a particular periodicity and the only way to extend them to time points outside that periodicity would be to construct a non-Markovian (and non-stationary) process.

If the periodicity of a discrete-time Markov chain is inherent to the process, then this result is not of concern. However, many systems do not have a natural sampling rate. The rate is chosen for computational or measurement convenience. In this case, we must be careful about how we employ our Markovian assumption. Or, we should directly model the underlying system in continuous time.

1.3.2 Independencies and Markovian-ness

A similar problem arises for independencies. We describe the problem here in terms of dynamic Bayesian networks (DBNs) (Dean & Kanazawa, 1989). If unfamiliar with DBNs, the reader may skip to the next section.

Consider the DBN in Figure 1(top,a), which has been unrolled one time step. We can marginalize out the middle time slice and the result is the DBN in Figure 1(top,b): the same model, but over twice the sampling periodicity. However, perhaps we wish to go the opposite direction (to half the sampling periodicity). Figure 1(bottom,b) shows a DBN over

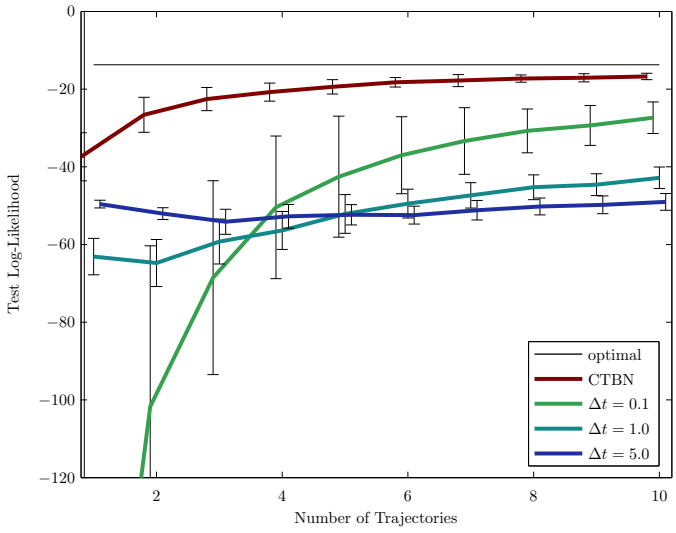


Figure 2: Comparison of learning DBNs with different time-slice widths

two time units. There is no DBN graph structure over one time unit that would marginalize to this graph structure. There may be a DBN, but the independencies expressed by the graph structure over two time units are not expressible in the *graph structure* at half the sampling periodicity. Such independencies would be buried in the parameters of the DBN (if those parameters are possible, given the previous discussion). This means that the independencies expressed by a DBN's graph are a function of both the underlying process and the sampling rate.

1.3.3 Structure Learning

Selection of a sampling rate is not just a theoretical problem. Nodelman, Shelton, and Koller (2003) demonstrate the problem for parameter estimation. In particular, they considered data drawn from a continuous-time Markovian process of eight variables (mostly binary). The resulting trajectories were discretized in time according to a parameter Δt and DBNs (including structure) were learned for each setting. Figure 2 shows the test log-likelihood accuracy as a function of the number of training trajectories and Δt. It also shows the result of not discretizing time (the CTBN line, a model explained in Section 5).

While it is not too surprising that the CTBN model does the best (as the data were generated from this model), it is instructive that the best Δt depends on the number of observed trajectories. This means that, if the sampling periodicity is a model parameter, its choice cannot be made independently of the amount of data used to estimate the DBN.

2. Continuous-Time Markov Processes

A continuous-time Markov process (CTMP) is a distribution over trajectories. A trajectory (or sample) of a CTMP is a right-continuous piece-wise constant function of a real-valued variable, time. Figure 3 illustrates example trajectories. If the states have a natural order-

(a) ordered states (b) unordered states (c) multiple state variables

Figure 3: Example continuous-time Markov process samples (trajectories)

ing, Figure 3(a) might be a natural depiction. If the states are not ordered, Figure 3(b) depicts the same sample for a three-state system. In later sections we will be considering large factored state spaces in which a state is an assignment to multiple variables. Figure 3(c) depicts such a trajectory.

A finite CTMP defines a set of random variables, each with the same finite sample space (the state space), indexed by a real-value usually denoted as t for time. Let X be such a process. The Markovian property states that

$$X(t_1) \perp X(t_3) \mid X(t_2), \forall\ t_1 < t_2 < t_3\ . \tag{5}$$

Throughout this tutorial, we will describe distributions over both continuous and discrete random variables. We will use lowercase letters for the densities of continuous random variables and uppercase letters for the probabilities of discrete random variables.

2.1 Parameterization

We will parameterize a CTMP X by a starting distribution at time $t = 0$, $P(X(0))$ (and restrict $t \geq 0$) and an intensity matrix $\boldsymbol{Q_X}$ (or just $\boldsymbol{Q}$ if the context is clear). The starting distribution is just as in a discrete-time Markov chain, and we will largely ignore it. The intensity matrix is analogous to the transition matrix of a discrete-time process.

2.1.1 Comparison to Discrete-Time

Consider the following (roughly equivalent) discrete-time transition matrix $\boldsymbol{M}$ and continuous-time intensity matrix $\boldsymbol{Q}$:

$$\boldsymbol{M} = \begin{bmatrix} 0.5 & 0.2 & 0.3 \\ 0.1 & 0.8 & 0.1 \\ 0.2 & 0.1 & 0.7 \end{bmatrix} \qquad \boldsymbol{Q} = \begin{bmatrix} -0.8 & 0.32 & 0.48 \\ 0.12 & -0.24 & 0.12 \\ 0.27 & 0.13 & -0.4 \end{bmatrix}\ .$$

We can interpret a row of $\boldsymbol{M}$ in two ways. The first row could be viewed as stating that if the process is in state 1, at the next time step there is a 0.5 chance that it will be in state 1, a 0.2 chance that it will be in state 2, and a 0.3 chance that it will be in state 3. Alternatively, it could be viewed as stating that if the process is in state 1, it will remain there for a number of steps geometrically distributed: Pr(stay for n steps) $= 0.5^n$. And, when it leaves it will transition to state 2 with probability $0.2/0.5 = 0.4$ and to state 3 with probability $0.3/0.5 = 0.6$.

The intensity matrix $\boldsymbol{Q}$ has two similar interpretations. The first row states that if the process is in state 1, after a short period of time ϵ, there is approximately a $1 - 0.8\epsilon$ chance of being in state 1, a 0.32ϵ chance of being in state 2, and a 0.48ϵ chance of being in state

3. The approximation has error $O(\epsilon^2)$. Alternatively, it states that if the process is in state 1 it remains there for a duration exponentially distributed: $p(\text{dwell time} = t) = 0.8e^{-0.8t}$. And when it leaves, it will transition to state 2 with probability $0.32/0.8 = 0.4$ and to state 3 with probability $0.48/0.8 = 0.6$.

2.1.2 Racing Exponentials

We can also view a row of the matrix as describing racing exponential distributions. There are two important properties of an exponential distribution. First, it is memoryless:

$$p_Z(t) = p_Z(t + s|Z > s) \qquad \text{if } Z \text{ is exponentially distributed} \tag{6}$$

and thus is the right distribution for dwell times in a Markovian process. (The amount of time the process has stayed in this state does not affect the remaining dwell time.) It is also closed under minimization: Given a collection of random variables $Z_1, Z_2, \ldots, Z_n$,

$$p_{Z_i}(t) = r_i e^{-r_i t} \tag{7}$$
$$Y = \min_i Z_i \tag{8}$$
$$J = \arg\min_i Z_i \tag{9}$$

implies

$$p_Y(t) = re^{-rt} \tag{10}$$
$$\Pr(J = j) = \frac{r_j}{r} \tag{11}$$

where

$$r = \sum_{i=1}^{n} r_i \ . \tag{12}$$

That is, if we have a set of exponential distributions with (potentially) different rates, their minimum (the time of the earliest one) is also exponentially distributed with rate equal to the sum of the component rates. Furthermore, the component which causes this minimum is independent of the time and is proportional to that component's rate. Thus, we can view each row of the matrix as a set of "racing" exponential distributions: one for each potential next state with rates dictated by the off-diagonal elements. Whichever potential transition happens first is the one actually taken by the process, and the rest are discarded.

2.1.3 Event Decomposition

Further, we can use this to build an interpretation of the summation of two intensity matrices. If $\boldsymbol{Q} = \boldsymbol{Q_1} + \boldsymbol{Q_2}$, where both $\boldsymbol{Q_1}$ and $\boldsymbol{Q_2}$ are valid intensity matrices, then the process of $\boldsymbol{Q}$ can be viewed as the combination of processes of $\boldsymbol{Q_1}$ and $\boldsymbol{Q_2}$ in which both sub-processes "race" to produce the next transition: For each current state, the two sub-processes have their own rate of transition for the possible new states. Whichever transition happens first switches the state and the joint process continues with the new

state. We can also view this as two different *event* types each with its own intensity matrix. The process of $\boldsymbol{Q}$ is the joint process of running both events at the same time, but throwing away (marginalizing out) the event types associated with the transitions, leaving only the transitions themselves.

2.1.4 Infinitesimal Rate Semantics

More formally, the dynamics of an n-state CTMP are described by a n-by-n intensity matrix $\boldsymbol{Q}$. The diagonal elements of $\boldsymbol{Q}$ are non-positive and the non-diagonal elements of $\boldsymbol{Q}$ are non-negative. The rows of $\boldsymbol{Q}$ sum to 0 (thus the diagonal elements are the negative row sums, if the diagonal element is excluded from the sum). We will denote the i, j element of $\boldsymbol{Q}$ as $q_{i,j}$. Further, for notational simplicity, we will let $q_i = -q_{i,i}$. That is, q_i is the rate of leaving state i, the absolute value of the corresponding diagonal element.

If we let $\boldsymbol{p}(t)$ be a row-vector of the marginal distribution of the process at time t, then the semantics of $\boldsymbol{Q}$ can be stated as

$$\boldsymbol{p}(t+\epsilon) = \boldsymbol{p}(t)(\boldsymbol{I} + \epsilon\boldsymbol{Q}) + o(\epsilon) \ . \tag{13}$$

This implies that

$$\boldsymbol{p}(t+\epsilon) - \boldsymbol{p}(t) = \epsilon\boldsymbol{p}(t)\boldsymbol{Q} + o(\epsilon) \tag{14}$$

$$\lim_{\epsilon\to 0}(\boldsymbol{p}(t+\epsilon) - \boldsymbol{p}(t))/\epsilon = \boldsymbol{p}(t)\boldsymbol{Q} \tag{15}$$

$$\frac{d\boldsymbol{p}(t)}{dt} = \boldsymbol{p}(t)\boldsymbol{Q} \tag{16}$$

$$\tag{17}$$

This first-order linear homogeneous differential equation has solution

$$\boldsymbol{p}(t+s) = \boldsymbol{p}(t)e^{\boldsymbol{Q}s} \tag{18}$$

assuming $s > 0$ and the initial conditions at t, $\boldsymbol{p}(t)$, are known. The exponential is the matrix exponential, defined by its Taylor expansion:

$$e^{\boldsymbol{Q}s} = \sum_{k=0}^{\infty} \frac{s^k}{k!}\boldsymbol{Q}^k \ . \tag{19}$$

Although not often practical computationally, we can also express the matrix exponential in terms of the eigenvalues ($\{\lambda_i\}$) and corresponding right and left eigenvectors ($\{\boldsymbol{v}_i\}$ and $\{\boldsymbol{w}_i\}$) of $\boldsymbol{Q}$:

$$e^{\boldsymbol{Q}s} = \sum_i e^{\lambda_i s}\boldsymbol{v}_i\boldsymbol{w}_i^\top \ . \tag{20}$$

If $\boldsymbol{Q}$ is of finite size and irreducible (there is a positive-rate path from any state to any other state), then the process is ergodic (the notion of "cycling" behavior does not exist in continuous-time Markov processes) and there will be exactly one eigenvalue equal to 0. The corresponding right eigenvector is the unique steady-state (or stationary) distribution of the process. If the process is not ergodic, then there *may* be multiple 0 eigenvalues and no unique stationary distribution. All other eigenvalues will be less than 0 and correspond to transients in the system. Therefore, $\boldsymbol{Q}$ will always be negative semi-definite.

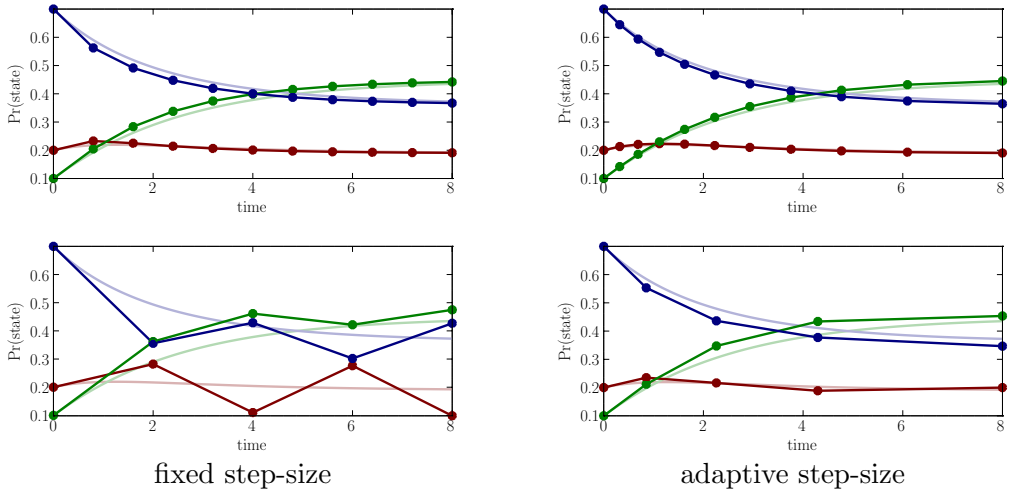


Figure 4: Propagation of marginal distribution from time 0 to time 8 by Euler integration. Left: fixed step-size. Right: adaptive step-size. Top: 11 evaluation points. Bottom: 5 evaluation points.

2.2 Matrix Exponential

The matrix exponential plays a critical role in many aspects of reasoning about continuous-time dynamic systems. At first, this would seem to be a significant downside, relative to discrete-time systems. Propagation of distribution $\boldsymbol{p}$ (as a vector) n time steps in a discrete-time system requires the multiplication by $\boldsymbol{M}^n$ (if $\boldsymbol{M}$ is the stochastic transition matrix). By contrast, the same operation in continuous-time requires the calculation of the matrix exponential, which is an infinite sum of matrix powers.

Consider computing the marginal distribution at time t by integrating the differential equation of Equation 18. The simplest method would be to use Euler integration with a fixed step size of Δt. This amounts to propagating over a fixed time interval by multiplying by $\boldsymbol{M} = \boldsymbol{I} + \Delta t \boldsymbol{Q}$. This is essentially the same as discretizing time and approximating the stochastic matrix over the resulting interval. To propagate to time t requires $t/\Delta t$ matrix multiplications. This is shown on the left side of Figure 4.

However, because time is continuous, we need not limit ourselves to time steps of uniform size. If we choose an adaptive step size, we can achieve the same accuracy with fewer evaluation points. Figure 4 demonstrates this for the simplest adaptive scheme (Euler steps with step size proportional to derivative magnitude). Note that for the same number of steps (computational complexity), the accuracy with adaptive steps is better.

For real applications, we would use a more advanced differential equation solver with more intelligent step size selection; see the work of Press, Teukolsky, Vetterling, and Flannery (1992) for an introduction. Yet, the idea is essentially the same: we can take larger steps during "less interesting" time periods. To do something similar with discrete time would require computations that essentially convert the discrete-time system to a continuous-

time one. Techniques like squaring and scaling to multiply by large matrix powers can also be applied to the matrix exponential. For a full discussion of matrix exponential calculations, we refer to the excellent treatments of Moler and Loan (2003) and Najfeld and Havel (1994, 1995).

2.3 Uniformization

Uniformization (also called randomization) is a method for converting questions about a continuous-time Markov process into ones about a discrete-time one (Grassmann, 1977). Given an intensity matrix $\boldsymbol{Q}$, uniformization constructs the stochastic matrix

$$\boldsymbol{M} = \boldsymbol{Q}/\alpha + \boldsymbol{I} \tag{21}$$

where $\alpha \geq \max_i q_i$ (that is, α is no less than the largest rate in $\boldsymbol{Q}$). For example, the following uniformization is with $\alpha = 0.5$ (the smallest possible value for α).

$$\boldsymbol{Q} = \begin{bmatrix} -0.5 & 0.1 & 0.4 \\ 0.1 & -0.2 & 0.1 \\ 0.2 & 0.1 & -0.3 \end{bmatrix} \rightarrow \boldsymbol{M} = \begin{bmatrix} 0.0 & 0.2 & 0.8 \\ 0.2 & 0.6 & 0.2 \\ 0.4 & 0.2 & 0.4 \end{bmatrix} \tag{22}$$

The resulting stochastic matrix can be interpreted as a discrete-time process. However it is *not* equivalent to sampling the continuous-time process at uniform intervals. Nor is it, in general, equivalent to the embedded Markov chain (the sequence of states, discarding the times of transitions). The former is achieved through the matrix exponential and the latter is achieved by setting all diagonal elements of $\boldsymbol{Q}$ to zero and then normalizing each row to sum to one.

Rather, the discrete-time process associated with the stochastic matrix $\boldsymbol{M}$ is related to the continuous-time process associated with the intensity matrix $\boldsymbol{Q}$ in the following way. Consider the procedure of (1) sampling event times from a Poisson process with rate α (that is, the times between consecutive events are independently and identically distributed as an exponential distribution with rate α), then (2) sampling state transitions at these event times from the discrete-time process described by $\boldsymbol{M}$, and then (3) discarding any self transitions. This procedure produces samples from the same distribution as the original CTMP described by $\boldsymbol{Q}$.

This transformation is useful for simulation (sampling), but also for understanding and computing the matrix exponential. Because the intensity matrix $\boldsymbol{Q}$ is negative semi-definite, the Taylor expansion of the matrix exponential is unstable, as the "sign" of the terms alternates. However, we can fix this by using $\boldsymbol{M}$ instead of $\boldsymbol{Q}$. By reworking Equation 21, we note that $\boldsymbol{Q} = \alpha(\boldsymbol{M} - \boldsymbol{I})$. We can then write

$$e^{\boldsymbol{Q}t} = e^{\alpha(\boldsymbol{M}-\boldsymbol{I})t} \tag{23}$$

$$= e^{-\alpha t} e^{\alpha \boldsymbol{M} t} \qquad [e^{\boldsymbol{A}+\boldsymbol{B}} = e^{\boldsymbol{A}} e^{\boldsymbol{B}} \text{ if } \boldsymbol{A}\boldsymbol{B} = \boldsymbol{B}\boldsymbol{A}] \tag{24}$$

$$= e^{-\alpha t} \sum_{k=0}^{\infty} \frac{\alpha^k t^k}{k!} \boldsymbol{M}^k \tag{25}$$

$$= \sum_{k=0}^{\infty} \underbrace{e^{-\alpha t} \frac{\alpha^k t^k}{k!}}_{\beta_k} \boldsymbol{M}^k \tag{26}$$

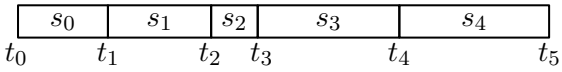


Figure 5: Pictorial representation of a finite-length sample from a CTMP.

where β_k is the probability of having exactly k events from a Poisson process with rate α in time t. This series is more stable ($\boldsymbol{M}$ is positive semi-definite) and for a finite number of terms, the sum is a quasi-probability vector (it is non-negative and sums to less than 1). The missing probability is a bound on the error. The sequence β_k grows and then decays. Therefore, discarding not only the tail of the series, but also early terms can speed up computations. Fox and Glynn (1988) give a method to compute left and right bounds on k to ensure a desired error tolerance.

Note that, if $\boldsymbol{Q}$ represents an ergodic continuous-time Markov process, then $\boldsymbol{M}$ represents an ergodic discrete-time Markov process when α is strictly greater than $\max_i q_i$ (a sufficient, but not necessary condition). If $\boldsymbol{M}$ is ergodic, then the stationary distribution of $\boldsymbol{Q}$ is the same as the stationary distribution of $\boldsymbol{M}$.

2.4 Likelihood

A complete finite-length sample (trajectory) $\mathcal{Tr}$ from a CTMP is sequence of states and the times of the transitions, plus an ending time: $\mathcal{Tr} = \{(s_0, t_0), (s_1, t_1), \ldots, (s_{n-1}, t_{n-1})\}, t_n$. Figure 5 shows a pictorial representation, for $n = 5$. If we use the convention that the process starts at time 0, then t_0 must be equal to 0.

The likelihood of this sample is the product of the conditional probabilities of each event (the starting state, each dwell duration, and each state transition):

$$p(\mathcal{Tr}) = \overbrace{\Pr(X(t_0) = s_0)}^{\text{init dist}} \prod_{i=0}^{n-2} \left(\overbrace{q_{s_i} e^{-q_{s_i}(t_{i+1}-t_i)}}^{\text{density of duration}} \overbrace{\frac{q_{s_i,s_{i+1}}}{q_{s_i}}}^{\text{pr of trans}} \right) \overbrace{e^{-q_{s_{n-1}}(t_n - t_{n-1})}}^{\text{pr of last duration}} \tag{27}$$

$$= P_0(s_0) \prod_{i=0}^{n-1} e^{-q_{s_i}(t_{i+1}-t_i)} \prod_{i=0}^{n-2} q_{s_i,s_{i+1}} \tag{28}$$

$$\ln p(\mathcal{Tr}) = \ln P_0(s_0) - \sum_{i=0}^{n-1} q_{s_i}(t_{i+1} - t_i) + \sum_{i=0}^{n-2} \ln q_{s_i,s_{i+1}} \tag{29}$$

We let P_0 be the distribution over the starting state of the process. Note that at time t_n the process does not transition. Rather, we observe that the process remains in state s_{n-1} for a duration of at least $t_n - t_{n-1}$.

Equation 29 can be rewritten as

$$\ln p(\mathcal{Tr}) = \ln P_0(s_0) - \sum_s T[s] q_s + \sum_{s \neq s'} N[s, s'] \ln q_{s,s'} \tag{30}$$

where $T[s]$ is the total time spent in state s and $N[s, s']$ is the total number of transitions from s to s', both of which are functions of $\mathcal{Tr}$. This demonstrates that a CTMP is a

member of an exponential family in which the sufficient statistics are $T[\cdot]$ and $N[\cdot,\cdot]$ (plus the relevant sufficient statistics for the starting distribution), and the natural parameters are the diagonal elements of the intensity matrix and the logarithm of the non-diagonal elements. The likelihood of multiple trajectories has the same form, where $T[s]$ and $N[s,s']$ are the sums of the sufficient statistics over the individual trajectories.

2.4.1 Parameter Estimation

The maximum likelihood parameters can be easily derived by differentiating Equation 30, after replacing q_s with $\sum_{s' \neq s} q_{s,s'}$:

$$\frac{\partial \ln p(\mathcal{T}r)}{\partial q_{s,s'}} = \frac{N[s,s']}{q_{s,s'}} - T[s] \qquad \forall\ s' \neq s \tag{31}$$

which implies the ML parameters are

$$\hat{q}_{s,s'} = N[s,s']/T[s] \qquad \forall\ s' \neq s \tag{32}$$

$$\hat{q}_s = \sum_{s' \neq s} N[s,s']/T[s] \qquad \forall\ s \tag{33}$$

A maximum a posteriori (MAP) estimate can be calculated if we place suitable prior distributions on the parameters. In particular, we will put an independent gamma distribution prior over each of the independent parameters, $q_{s,s'}, \forall\ s \neq s'$:

$$p(q_{s,s'}; \alpha_{s,s'}, \tau_{s,s'}) = \frac{\tau_{s,s'}^{\alpha_{s,s'}+1}}{\Gamma(\alpha_{s,s'}+1)} q_{s,s'}^{\alpha_{s,s'}} e^{-q_{s,s'}\tau_{s,s'}} \tag{34}$$

which has parameters $\alpha_{s,s'}$ and $\tau_{s,s'}$. The posterior distribution over the parameters given data summarized by the sufficient statistics $T[s]$ and $N[s,s']$ is also gamma-distributed with parameters $\alpha_{s,s'} + N[s,s']$ and $\tau_{s,s'} + T[s]$. Thus, the MAP estimates of the parameters are

$$\hat{q}_s = \sum_{s'} \frac{N[s,s'] + \alpha_{s,s'}}{T[s] + \tau_{s,s'}} \tag{35}$$

$$\hat{q}_{s,s'} = \frac{N[s,s'] + \alpha_{s,s'}}{T[s] + \tau_{s,s'}}\ . \tag{36}$$

2.5 Inference

We will now consider two classic problems of reasoning in temporal systems: filtering and smoothing. Initially, we will assume we have observations (evidence) with a pattern like that in Figure 6: a sequence of times $\{t_0, t_1, \ldots, t_k\}$ and a sequence of evidences $\{e_1, e_2, \ldots, e_k\}$. We assume that we know the prior marginal distribution over $X(t_0)$, either from previous reasoning or because $t_0 = 0$.

Filtering is the task of computing $p(X(t) \mid e_1, e_2, \ldots, e_k)$ for $t \geq t_k$. If the evidence at each point is an observation of the state of the system, the Markovian property of the process makes inference trivial. Instead we will assume that e_i is only probabilistically related to $X(t_i)$ but independent of everything else given $X(t_i)$. (This is analogous to a discrete-time

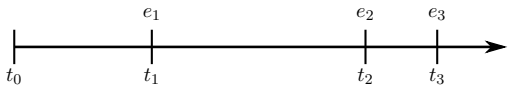


Figure 6: Example evidence pattern, point evidence.

hidden Markov model.) Thus we can view each observation as a noisy measurement of the system.

As with a hidden Markov model, we define a recursive filtering solution using a forward "message" $\boldsymbol{\alpha}$ whose components are defined as

$$\boldsymbol{\alpha}_i(t) = \Pr\left(X(t) = i, e_{[t_0,t)}\right) \tag{37}$$

where we denote $e_{[s,t)} = \{(t_i, e_i) \mid s \leq t_i < t\}$: the set of evidence in the interval $[s,t)$. By analogy we will also define $e_{[s,t]}$ and $e_{(s,t]}$ to be the evidence on $[s,t]$ and $(s,t]$ respectively (which we will need later). Note that $\boldsymbol{\alpha}$ is a *row* vector of probabilities, one for each state of the system. Recursive calculation of $\boldsymbol{\alpha}$ can be derived from

$$\Pr\left(X(t) = j, e_{[t_0,t)}\right) = \sum_i \Pr\left(X(s) = i, e_{[t_0,s)}\right) \Pr\left(X(t) = j, e_{[s,t)} \mid X(s) = i\right) \quad \forall\ t_0 \leq s < t \tag{38}$$

$$\boldsymbol{\alpha}(t) = \boldsymbol{\alpha}(s)\boldsymbol{F}(s,t) \qquad \forall\ t_0 \leq s < t \tag{39}$$

where the second equation is the vector version of the first equation, and the matrix $\boldsymbol{F}(s,t)$ has element i,j equal to $\Pr\left(X(t) = j, e_{[s,t)} \mid X(s) = i\right)$.

If there is no evidence in $[s,t)$, $\boldsymbol{F}(s,t) = e^{\boldsymbol{Q}(t-s)}$. Thus, we can propagate the distribution from one evidence time point to the next with the matrix exponential. To propagate across evidence times, we define

$$\boldsymbol{\alpha}^+(t) = \Pr\left(X(t), e_{[t_0,t]}\right) \tag{40}$$

to be the same as $\boldsymbol{\alpha}(t)$, but including evidence at t. If there is no evidence at t, the two vectors are the same. If there is evidence at t, then $\boldsymbol{\alpha}^+(t)$ is the same as $\boldsymbol{\alpha}(t)$, except that each element is multiplied by the probability of the evidence at that time point.

If we let $\boldsymbol{O^{(i)}}$ be a diagonal matrix in which diagonal element j is $\Pr(e_i \mid X(t_i) = j)$, then the recurrence for $\boldsymbol{\alpha}$ can be written as

$$\boldsymbol{\alpha}(t_0) = \boldsymbol{\alpha}^+(t_0) = \text{given} \tag{41}$$

$$\boldsymbol{\alpha}(t_i) = \boldsymbol{\alpha}^+(t_{i-1})e^{\boldsymbol{Q}(t_i - t_{i-1})} \qquad \forall\ 0 < i \leq k \tag{42}$$

$$\boldsymbol{\alpha}^+(t_i) = \boldsymbol{\alpha}(t_i)\boldsymbol{O^{(i)}} \qquad \forall\ 0 < i \leq k \tag{43}$$

$$\boldsymbol{\alpha}(t) = \boldsymbol{\alpha}^+(t_i)e^{\boldsymbol{Q}(t - t_i)} \qquad \forall\ t_i < t \leq t_{i+1} \text{ or } t_i < t, i = k \tag{44}$$

Equation 42 is a special case of Equation 44. It propagates from "just after" one evidence time until "just before" the next. Equation 43 propagates "across" an evidence point.

Equation 44 can be used to construct the filtered estimate at any non-evidence time by normalizing $\boldsymbol{\alpha}(t)$ to sum to 1 (dividing by the probability of the evidence prior to t).

Finally, note that a similar set of recurrences can be derived for $F(\cdot,\cdot)$. The result allows for the propagation of any distribution across time intervals which includes evidence; that is, we are not restricted to any particular initial condition, $\boldsymbol{\alpha}(t_0)$. However, if only a single $\boldsymbol{\alpha}$ is to be propagated, computing F first is more computationally expensive.

2.5.1 More Complex Evidence

The above filtering equations are just an inhomogeneous hidden Markov model (that is, the transition matrix is not constant) and familiar to those who have employed hidden Markov models. However, with continuous time, there are evidence patterns that do not have direct corresponding analogies in discrete-time. If the evidence consists of a finite number of observations, we can convert it into a similar form, by breaking time into intervals of constant evidence.

For instance, we might observe that the system is in a subset of states for a duration of time: During this time interval, the system does not leave the subset, but we do not observe whether there are any transitions within the subset. We can augment our evidence to include this information. For each interval $[t_{i-1}, t_i)$, we let S_i denote the subset of states in which the evidence constrains the system. If there are no such constraints, S_i is the full state space. For time points at which there is a change in S_i, but no point evidence, $\boldsymbol{O^{(i)}}$ is the identity matrix (inducing no change in the filtering estimate). Both S_i and $\boldsymbol{O^{(i)}}$ may be non-trivial for the same i.

Now to propagate from t_{i-1} to t_i, we must use a modified intensity matrix. In particular, we set to zero any rate which is inconsistent with the evidence S_i: all rates to, from, or within the set of states that are *not* in S_i. Let $\boldsymbol{Q^{(i)}}$ denote such a matrix. If the rows and columns are permuted such that the states in S_i are in the upper left corner, then this matrix has the form

$$\boldsymbol{Q^{(i)}} = \begin{bmatrix} \tilde{\boldsymbol{Q}}_{\boldsymbol{S_i}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{45}$$

where $\tilde{\boldsymbol{Q}}_{\boldsymbol{S_i}}$ is the submatrix of $\boldsymbol{Q}$ of the rows and columns corresponding to S_i. Additionally, we modify $\boldsymbol{O^{(i-1)}}$, setting to 0 any diagonal elements corresponding to states *not* in S_i.

Note that $\boldsymbol{Q^{(i)}}$ is not (strictly) an intensity matrix: its rows do not sum to 0. In general, a diagonal element is greater (in absolute value) than the sum of the other row elements because we have set to zero non-diagonal rates. This "missing rate" corresponds to the probability of leaving the evidence set (and therefore not conforming to the evidence). While $e^{\boldsymbol{Q}(t_i - t_{i-1})}$ is a stochastic matrix representing the conditional distribution at time t_i given the state at time t_{i-1}, $e^{\boldsymbol{Q^{(i)}}(t_i - t_{i-1})}$ is a substochastic matrix (the row sums are less than or equal to 1), where the sum of each row is the probability of the evidence over the interval, given the state at time t_{i-1}.

The new filtering recurrence is

$$\boldsymbol{\alpha}(t_0) = \text{given} \tag{46}$$

$$\boldsymbol{\alpha}(t_i) = \boldsymbol{\alpha}^+(t_{i-1}) e^{\boldsymbol{Q^{(i)}}(t_i - t_{i-1})} \qquad \forall\ 0 < i \leq k \tag{47}$$

$$\boldsymbol{\alpha}^+(t_i) = \boldsymbol{\alpha}(t_i)\boldsymbol{O^{(i)}} \qquad \forall\ 0 < i \leq k \tag{48}$$

$$\boldsymbol{\alpha}(t) = \boldsymbol{\alpha}^+(t_i) e^{\boldsymbol{Q^{(i+1)}}(t - t_i)} \qquad \forall\ t_i < t \leq t_{i+1} \text{ or } t_i < t, i = k\ . \tag{49}$$

We might also observe a transition at an exact time point. More generally, at time t_i we might observe that a transition occurred from one state of the set U_i^- to one state of the set U_i^+ (without knowing exactly which states within the sets). In this case, elements of $\boldsymbol{\alpha}(t)$ have the probabilities of a duration lasting until at least t, and $\boldsymbol{\alpha}^+(t)$ should have the probability density of a duration lasting exactly until t. The difference between the probability of the tail of an exponential and the density at the same point is just a multiplication by the relevant rate q. Thus, for this type of evidence, we can just modify $\boldsymbol{O^{(i)}}$. In particular,

$$\boldsymbol{O^{(i)}}_{j,k} = \begin{cases} q_{j,k} & \text{if } \ j \in U_i^-,\ k \in U_i^+, \text{ and } j \neq k \\ 0 & \text{otherwise} \end{cases}\ . \tag{50}$$

The recurrence remains the same, with the new definition of $\boldsymbol{O^{(i)}}$. Other evidence types are also possible and can be derived from the above types by augmenting the state space.

2.5.2 Smoothing

Smoothing is the problem of calculating $\Pr\left(X(t) \mid e_{[t_0,t_k]}\right)$ for $t_0 \leq t \leq t_k$. As common with Markov processes, we note that

$$\Pr\left(X(t) \mid e_{[t_0,t_k]}\right) \propto \Pr\left(X(t) \mid e_{[t_0,t)}\right) \Pr\left(e_{[t,t_k]} \mid X(t)\right) \tag{51}$$

where the constant of proportionality can be found by noting that the sum of Equation 51 over the value of $X(t)$ must equal 1. The first term on the right is calculated with the $\boldsymbol{\alpha}(\cdot)$ recurrence above. The second term we calculate with a backward message recurrence. Define

$$\boldsymbol{\beta}_i(t) = \Pr\left(e_{[t,t_k]} \mid X(t) = i\right) \tag{52}$$

$$\boldsymbol{\beta}_i^+(t) = \Pr\left(e_{(t,t_k]} \mid X(t) = i\right) \tag{53}$$

If we let $\boldsymbol{\beta}$ be a *column* vector, then the backward recurrence is analogous the forward one, but with right multiplication instead of left multiplication:

$$\boldsymbol{\beta}^+(t_k) = \mathbf{1} \qquad \text{vector of 1s} \tag{54}$$

$$\boldsymbol{\beta}(t_i) = \boldsymbol{O^{(i)}}\boldsymbol{\beta}^+(t_i) \qquad \forall\ 0 < i \leq k \tag{55}$$

$$\boldsymbol{\beta}^+(t_{i-1}) = e^{\boldsymbol{Q^{(i)}}(t_i - t_{i-1})}\boldsymbol{\beta}(t_i) \qquad \forall\ 0 < i \leq k \tag{56}$$

$$\boldsymbol{\beta}(t) = e^{\boldsymbol{Q^{(i)}}(t_i - t)}\boldsymbol{\beta}(t_i) \qquad \forall\ t_{i-1} \leq t < t_i\ . \tag{57}$$

For any time t, the vector of the distribution of the state of the system at t given all the evidence is

$$\boldsymbol{p}(X(t) \mid e_{[t_0,t_k]}) \propto \boldsymbol{\alpha}(t) \odot \boldsymbol{\beta}(t) \tag{58}$$

where $\odot$ is the Hadamard (point-wise) product.

2.6 Parameter Estimation from Incomplete Evidence

Section 2.4.1 demonstrated that a CTMP is a member of an exponential family with sufficient statistics $T[i]$ (the amount of time spent in state i) and $N[i,j]$ (the number of transitions from i to j). If the evidence trajectories are fully observed over a continuous interval of time, then these sufficient statistics can be trivially tallied. Further, if each evidence trajectory is observed at $t = 0$, the sufficient statistics for the initial distribution are also directly observed.

However, if portions of the interval are hidden, or more generally the observations are of the form of the previous section, direct likelihood maximization is not feasible. There are two basic approaches for maximum likelihood estimation in this case: gradient ascent and expectation maximization (EM).

For gradient ascent, we can replace the sufficient statistics in Equation 31 with their expected values The standard argument for exponential models applies: Let $\mathcal{Tr}$ be a partially observed trajectory and let h stand for any potential completion of it.

$$\ln p(\mathcal{Tr}) = \ln \sum_h e^{\ln p(\mathcal{Tr},h)} \tag{59}$$

$$\frac{\partial \ln p(\mathcal{Tr})}{\partial q_{i,j}} = \frac{1}{p(\mathcal{Tr})} \sum_h p(\mathcal{Tr},h) \frac{\partial \ln p(\mathcal{Tr},h)}{\partial q_{i,j}} \tag{60}$$

$$= E_{h|\mathcal{Tr}} \left[\frac{N[i,j]}{q_{i,j}} - T[i] \right] \tag{61}$$

$$= \left(\frac{\bar{N}[i,j]}{q_{i,j}} - \bar{T}[i] \right) \tag{62}$$

where $\bar{N}[i,j]$ and $\bar{T}[i]$ are the expected values of $N[i,j]$ and $T[i]$ with respect to completions of $\mathcal{Tr}$. For EM, we similarly replace $N[i,j]$ and $T[i]$ in Equation 32 with $\bar{N}[i,j]$ and $\bar{T}[i]$. We are therefore left with the problem of computing the expected values of $N[i,j]$ and $T[i]$.

Full derivations are shown in the work of Nodelman et al. (2003). A quick version for $\bar{T}[i]$ is

$$\bar{T}[i] = \int_{t_0}^{t_k} p(X(t) = i \mid e_{[t_0,t_k]})\ dt \tag{63}$$

$$= \frac{1}{p(\mathcal{Tr})} \int_{t_0}^{t_k} \boldsymbol{\alpha}_i(t) \boldsymbol{\beta}_i(t)\ dt\ . \tag{64}$$

The expected value of $N[i,j]$ has a similar form:

$$\bar{N}[i,j] = \frac{q_{i,j}}{p(\mathcal{Tr})} \int_{t_0}^{t_k} \boldsymbol{\alpha}_i(t) \boldsymbol{\beta}_j(t)\ dt + \sum_{l \in \text{Trans}} \frac{\boldsymbol{\alpha}_i(t_l) \boldsymbol{O^{(l)}}_{i,j} \boldsymbol{\beta}_j^+(t_l)}{\sum_{i',j'} \boldsymbol{O^{(l)}}_{i',j'}} \tag{65}$$

where Trans is the set of evidence indices at which time a transition was (perhaps partially) observed: The first term handles unobserved transitions and the second handles (partially) observed transitions.

If we let $\boldsymbol{\Delta_{i,j}}$ be a matrix of all zeros, except for a single one in location (i, j), the integrals in both Equation 64 and Equation 65 have the form

$$\int_{t_0}^{t_k} \boldsymbol{\alpha}_i(t)\boldsymbol{\beta}_j(t)\ dt = \sum_{l=1}^{k} \int_{t_{l-1}}^{t_l} \boldsymbol{\alpha}_i(t)\boldsymbol{\beta}_j(t)\ dt \tag{66}$$

$$= \sum_{l=1}^{k} \int_{t_{l-1}}^{t_l} \boldsymbol{\alpha}^+(t_{l-1}) e^{\boldsymbol{Q^{(i)}}(t-t_{l-1})} \boldsymbol{\Delta_{i,j}} e^{\boldsymbol{Q^{(i)}}(t_l-t)} \boldsymbol{\beta}(t_l)\ dt\ . \tag{67}$$

Thus, after a forward and backward pass to calculate $\boldsymbol{\alpha}^+(i)$ and $\boldsymbol{\beta}(i)$ at each evidence change point i, each integral is relatively simple. They can be solved by standard quadrature methods or by the solution of a differential equation (Asmussen, Nerman, & Olsson, 1996). Alternatively, the calculation of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ usually results in values for each at various time points, which can be interpolated to full functions and used to directly solve the integrals.

3. Kronecker Algebra Representations

When the number of states is no more than a few thousand, the above methods are computationally feasible on a modern computer. However, most models are described in terms of assignments to variables. Thus the number of states grows exponentially with the number of variables. For more than a few tens of variables, we must seek more compact representations.

For the remainder of this paper, we will consider the state space of the process X to be an assignment to L variables, $\{X_1, X_2, \ldots, X_L\}$. We let variable X_i have n_i possible assignments. Thus, the total state space is of size $n = \prod_{i=1}^{L} n_l$. We let a bold $\boldsymbol{x}$ stand for a state (joint assignment to all L variables), with component x_i being the assignment to variable i in state $\boldsymbol{x}$. Such a state space is often referred to as *factored* or *structured* or *variable-based*.

Kronecker products and sums are natural "basic operations" from which to build compact representations of the process intensities. In some cases, these compact representations naturally describe the transition rates, but do not as naturally describe the diagonal elements of $\boldsymbol{Q}$ (the negative rates of leaving each state). Thus, we will define $\boldsymbol{R}$ to be the same as $\boldsymbol{Q}$, except with zeros at each diagonal position. The diagonals can be reconstructed from the non-diagonal elements in the same row, so the information content is the same.

3.1 Kronecker Product

The first basic operation is the Kronecker product. Given matrices $\boldsymbol{A^{(1)}}, \boldsymbol{A^{(2)}}, \ldots, \boldsymbol{A^{(K)}}$ where $\boldsymbol{A^{(k)}}$ is of general size m_k-by-n_k, the Kronecker product is written

$$\boldsymbol{A} = \bigotimes_{k=1}^{K} \boldsymbol{A^{(k)}} \tag{68}$$

where $\boldsymbol{A}$ (the result) is an m-by-n matrix: $m = \prod_k m_k$ and $n = \prod_k n_k$. The elements of $\boldsymbol{A}$ represent all possible multiplications of one element from each of $\boldsymbol{A^{(1)}}, \boldsymbol{A^{(2)}}, \ldots, \boldsymbol{A^{(K)}}$. Let $\mathcal{M}_k = \{1, 2, \ldots, m_k\}$ and $\mathcal{N}_k = \{1, 2, \ldots, n_k\}$, that is the valid indices into matrix $\boldsymbol{A^{(k)}}$.

$$\text{Given } \boldsymbol{A} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \quad \text{and} \quad \boldsymbol{B} = \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix}$$

$$\boldsymbol{A} \otimes \boldsymbol{B} = \left[\begin{array}{c|c} a_{00}\boldsymbol{B} & a_{01}\boldsymbol{B} \\ \hline a_{10}\boldsymbol{B} & a_{11}\boldsymbol{B} \end{array}\right] = \left[\begin{array}{ccc|ccc} a_{00}b_{00} & a_{00}b_{01} & a_{00}b_{02} & a_{01}b_{00} & a_{01}b_{01} & a_{01}b_{02} \\ a_{00}b_{10} & a_{00}b_{11} & a_{00}b_{12} & a_{01}b_{10} & a_{01}b_{11} & a_{01}b_{12} \\ a_{00}b_{20} & a_{00}b_{21} & a_{00}b_{22} & a_{01}b_{20} & a_{01}b_{21} & a_{01}b_{22} \\ \hline a_{10}b_{00} & a_{10}b_{01} & a_{10}b_{02} & a_{11}b_{00} & a_{11}b_{01} & a_{11}b_{02} \\ a_{10}b_{10} & a_{10}b_{11} & a_{10}b_{12} & a_{11}b_{10} & a_{11}b_{11} & a_{11}b_{12} \\ a_{10}b_{20} & a_{10}b_{21} & a_{10}b_{22} & a_{11}b_{20} & a_{11}b_{21} & a_{11}b_{22} \end{array}\right]$$

Figure 7: Example Kronecker product

Then, let $\mathcal{I}_r$ be a mapping from $\mathcal{M}_1 \times \mathcal{M}_2 \times \cdots \times \mathcal{M}_K$ to $\{1, 2, \ldots, m\}$ and let $\mathcal{I}_c$ be similarly defined as a mapping from $\mathcal{N}_1 \times \mathcal{N}_2 \times \cdots \times \mathcal{N}_K$ to $\{1, 2, \ldots, n\}$. It does not matter usually what the mappings are, but by convention we take them to be lexicographic orderings (or mixed-base numbering index). For instance $\mathcal{I}_r(i_1, i_2, \ldots, i_K) = \sum_{1 \le k \le K} i_k m_{1:k-1}$ where $m_{a:b} = \prod_{a \le k \le b} m_k$. Then

$$\boldsymbol{A}_{\mathcal{I}_r(i_1,i_2,\ldots,i_K),\mathcal{I}_c(j_1,j_2,\ldots,j_K)} = \prod_{k=1}^{K} \boldsymbol{A^{(k)}}_{i_k,j_k} \ . \tag{69}$$

While the notation makes it appear complex, the concept is simple. Figure 7 demonstrates a simple example. In terms of sparsity (one measure of structure), the Kronecker product has a number of non-zero elements equal to the product of the number of non-zero elements in each input matrix.

A Kronecker product is analogous to a factor product (in Bayesian network terminology) if we treat each operand matrix as a factor over two different variables (and no matrices share the same variables), and the result matrix is a factor in which half of the variables are flattened into the "column" dimension and the other half are flattened into the "row" dimension.

In terms of distributions, the Kronecker product represents *independence*. Given two variables X_1 and X_2 with marginal distributions represented by the vectors $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$, $\boldsymbol{v_1} \otimes \boldsymbol{v_2}$ is a joint distribution over both X_1 and X_2. In particular, it is the independent joint distribution with marginals $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$.

In terms of a rate matrix, the Kronecker product represents *synchronization* (Plateau, 1985). If we have two variables, X_1 and X_2 with rate[2] matrices $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$, $\boldsymbol{R_1} \otimes \boldsymbol{R_2}$ is a rate matrix over the state space $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2$ (joint assignments to X_1 and X_2). It represents a rate matrix in which changes in the state of X_1 must occur at the same time as those in the state of X_2 (both variables will be changed by every transition).

2. This does not hold generally for intensity matrices, as the Kronecker product does not do anything sensible with the diagonal elements.

$$\boldsymbol{A} \oplus \boldsymbol{B} = \boldsymbol{A} \otimes \boldsymbol{I_3} + \boldsymbol{I_2} \otimes \boldsymbol{B} =$$

$$\left[\begin{array}{ccc|ccc} a_{0,0} & & & a_{0,1} & & \\ & a_{0,0} & & & a_{0,1} & \\ & & a_{0,0} & & & a_{0,1} \\ \hline a_{1,0} & & & a_{1,1} & & \\ & a_{1,0} & & & a_{1,1} & \\ & & a_{1,0} & & & a_{1,1} \end{array}\right] + \left[\begin{array}{ccc|ccc} b_{0,0} & b_{0,1} & b_{0,2} & & & \\ b_{1,0} & b_{1,1} & b_{1,2} & & & \\ b_{2,0} & b_{2,1} & b_{2,2} & & & \\ \hline & & & b_{0,0} & b_{0,1} & b_{0,2} \\ & & & b_{1,0} & b_{1,1} & b_{1,2} \\ & & & b_{2,0} & b_{2,1} & b_{2,2} \end{array}\right] =$$

$$\left[\begin{array}{ccc|ccc} a_{0,0}{+}b_{0,0} & b_{0,1} & b_{0,2} & a_{0,1} & & \\ b_{1,0} & a_{0,0}{+}b_{1,1} & b_{1,2} & & a_{0,1} & \\ b_{2,0} & b_{2,1} & a_{0,0}{+}b_{2,2} & & & a_{0,1} \\ \hline a_{1,0} & & & a_{1,1}{+}b_{0,0} & b_{0,1} & b_{0,2} \\ & a_{1,0} & & b_{1,0} & a_{1,1}{+}b_{1,1} & b_{1,2} \\ & & a_{1,0} & b_{2,0} & b_{2,1} & a_{1,1}{+}b_{2,2} \end{array}\right]$$

Figure 8: Example Kronecker sum, given same matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ as in Figure 7. Zeros are omitted. Note that the non-zero off-diagonal entries all correspond to only one of the two indices (into $\boldsymbol{A}$ or $\boldsymbol{B}$) changing.

3.2 Kronecker Sum

The other Kronecker operation is the Kronecker sum. It is only defined on square matrices. Given square matrices $\boldsymbol{A^{(1)}}, \boldsymbol{A^{(2)}}, \ldots, \boldsymbol{A^{(K)}}$ where $\boldsymbol{A^{(k)}}$ has size n_k-by-n_k, the Kronecker sum is defined in terms of the Kronecker product:

$$\boldsymbol{A} = \bigoplus_{k=1}^{K} \boldsymbol{A^{(k)}} = \sum_{k=1}^{K} \boldsymbol{I_{n_1}} \otimes \boldsymbol{I_{n_2}} \otimes \ldots \boldsymbol{I_{n_{k-1}}} \otimes \boldsymbol{A^{(k)}} \otimes \boldsymbol{I_{n_{k+1}}} \otimes \cdots \otimes \boldsymbol{I_{n_K}} \tag{70}$$

where $\boldsymbol{I_n}$ is the identity matrix of size n-by-n. The Kronecker sum has the same size as the Kronecker product of the same matrices and we can use the same indexing function to reference elements in the sum, but we need only one because the matrix is square, thus $\mathcal{I}_r = \mathcal{I}_c = \mathcal{I}$:

$$A_{\mathcal{I}(i_1,i_2,\ldots,i_K),\mathcal{I}(j_1,j_2,\ldots,j_K)} = \begin{cases} \sum_{k=1}^{K} \boldsymbol{A^{(k)}}_{i_k,i_k} & \text{if } i_l = j_l \text{ for all } l \\ \boldsymbol{A^{(k)}}_{i_k,j_k} & \text{if } i_l = j_l \text{ for all } l \text{ except } l = k \\ 0 & \text{otherwise.} \end{cases} \tag{71}$$

Figure 8 demonstrates a simple example.

In terms of a CTMP, the Kronecker sum represents asynchronicity. Given two variables, X_1 and X_2 with intensity[3] matrices $\boldsymbol{Q_1}$ and $\boldsymbol{Q_2}$, $\boldsymbol{Q_1} \oplus \boldsymbol{Q_2}$ is an intensity matrix over the joint state space in which each process's events proceed irrespective of the other's state. That is, the processes are independent (assuming their starting distributions are independent).

3. The same holds for rate matrices, but we can be stronger here than for the Kronecker product and make this statement about the intensity matrices too.

Note that the intensity of any transition that involves two or more variables is zero (at any instant, a maximum of one variable can change).

3.3 Properties

The Kronecker product obeys the classic distributive property:

$$(\boldsymbol{A} + \boldsymbol{B}) \otimes \boldsymbol{C} = \boldsymbol{A} \otimes \boldsymbol{C} + \boldsymbol{B} \otimes \boldsymbol{C}$$

The mixed product property provides a relationship between the Kronecker product and the matrix product. Given matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$ and assuming that $\boldsymbol{AC}$ and $\boldsymbol{BD}$ are valid matrix products,

$$(\boldsymbol{A} \otimes \boldsymbol{B})(\boldsymbol{C} \otimes \boldsymbol{D}) = (\boldsymbol{AC}) \otimes (\boldsymbol{BD}) \ . \tag{72}$$

One consequence is that the Kronecker product can be expressed as

$$\bigotimes_{k=1}^{K} \boldsymbol{A^{(k)}} = \prod_{k=1}^{K} \boldsymbol{I_{n_1}} \otimes \boldsymbol{I_{n_2}} \otimes \cdots \otimes \boldsymbol{I_{n_{k-1}}} \otimes \boldsymbol{A^{(k)}} \otimes \boldsymbol{I_{n_{k+1}}} \otimes \cdots \otimes \boldsymbol{I_{n_K}} \tag{73}$$

$$= \prod_{k=1}^{K} \boldsymbol{I_{n_{1:k-1}}} \otimes \boldsymbol{A^{(k)}} \otimes \boldsymbol{I_{n_{k+1:K}}} \tag{74}$$

where $n_{a:b}$ is the product of the terms n_a through n_b as defined above. This shows a bit of the relationship between Kronecker products and sums: Compare Equation 70 and Equation 73.

This can be further reworked as

$$\bigotimes_{i=k}^{K} \boldsymbol{A^{(k)}} = \prod_{i=k}^{K} \boldsymbol{P_{n_{1:k},n_{k+1:K}}}^{\top} \cdot (\boldsymbol{I_{\overline{n}_k}} \otimes \boldsymbol{A^{(k)}}) \cdot \boldsymbol{P_{n_{1:k},n_{k+1:K}}} \tag{75}$$

where $\overline{n}_k = n_{1:K}/n_k$ and $\boldsymbol{P_{a,b}}$ is the matrix describing an a,b-perfect shuffle permutation of $(0, ..., ab-1)$: its entry in position (i, j) is 1 if $j = (i \bmod a) \cdot b + \lfloor i/a \rfloor$, and 0 otherwise (in particular, $\boldsymbol{P_{a,b}} = \boldsymbol{P_{a,b}} = \boldsymbol{I_{a \cdot b}}$ if a or b is 1). Whereas Equation 74 orders the Kronecker products of the outer product's terms so that the elements of $\boldsymbol{A_k}$ are in the correct places, Equation 75 repeats $\boldsymbol{A_k}$ on the diagonal and then permutes the rows and columns to place the elements in the correct locations. A similar transformation can be used to rewrite Equation 70 as a sum of shuffled block-diagonal matrices. Because the permutations can often be done implicitly in code, these versions can be useful in deriving algorithms.

3.4 Compact Kronecker Representations

Given a factored state space as before, any joint rate matrix $\boldsymbol{R}$ can be expressed as a sum of Kronecker products:

$$\boldsymbol{R} = \sum_{e=1}^{E} \bigotimes_{l=1}^{L} \boldsymbol{R_e^{(l)}} \tag{76}$$

where there are L variables and $\boldsymbol{R_e^{(l)}}$ is a rate matrix over the space of variable l only. In particular, an exponentially sized (in the number of variables) representation is straightforward: e ranges over the elements in the resulting matrix. For element corresponding to

$(x_1, x_2, \ldots, x_L), (x'_1, x'_2, \ldots, x'_L)$, $\boldsymbol{R}_e^{(l)} = \boldsymbol{\Delta}_{\boldsymbol{x}_l, \boldsymbol{x}'_l}$ for $1 < l \leq L$ and the same for $l = 1$, except multiplied by the scalar value to be placed in this location. In this way each term in the sum is a matrix with at most a single non-zero element. However, for many processes we can expect E to be a manageable number. For instance, if the variables are all independent, $E = L$ (and all but L of the L^2 rate components are identity matrices), as per the Kronecker sum above.

We can view each of the E terms in the sum as separate "events" whose identities have been marginalized out to produce the resulting process (see Section 2.1.3). These events must couple variables synchronously (due to the Kronecker product). We exploit this type of decomposition more extensively in the next sections.

4. Decision Diagram Representations

While the encoding in Equation 76 can be efficient, we can do better by exploiting more internal structure. $\boldsymbol{R}$ can be viewed as a mapping from two discrete domains (the row index and the column index) to a real value. *Decision diagrams* have long been used in computer science to compactly encode functions over discrete domains. Here we show how they have been used in CTMPs and how they can be seen as an alternative to Kronecker algebra encodings, in the case of the MTBDDs used in PRISM (Kwiatkowska, Norman, & Parker, 2011), or even as an extension of Kronecker algebra encodings, in the case of the Matrix Diagrams used in Möbius (Deavours, Clark, Courtney, Daly, Derisavi, Doyle, Sanders, & Webster, 2002) or the EV*MDDs used in SMART (Ciardo, Jones, Miner, & Siminiceanu, 2006).

4.1 Decision Diagram Overview

Decision diagrams encode functions of the form $f : X \to X_0$ where, as before, the *domain* state space X is structured: $X = X_1 \times \cdots \times X_L$. In other words, f is applied to a (state) tuple and evaluates to an element of a *range* set X_0. One can then think of f as the encoding of a vector indexed by X and having entries with values in X_0. Of course, the same idea can be employed to encode matrices, we simply need to use the domain $X \times X$. (In practice, we actually use the *interleaved* domain $X'_1 \times X_1 \times \cdots \times X'_L \times X_L$, where the "unprimed" state variables refer to row indices, or "from" states, while the "primed" state variable refer to column indices, or "to" states, as this usually leads to more compact decision diagrams.)

Binary decision diagrams, or BDDs (Bryant, 1986), encode functions for which all sets forming the domain X are binary, while multiway decision diagrams, or MDDs (Kam, Villa, Brayton, & Sangiovanni-Vincentelli, 1998), allow non-binary domain sets. However, for both the range X_0 is binary. For our numeric application, we need to extend such representations to allow the range X_0 to be either the integers $\mathbb{Z}$ (possibly augmented with the value ∞ to indicate "undefined") or the reals $\mathbb{R}$ (possibly, again, augmented with ∞, or restricted to the nonnegative reals $\mathbb{R}^{\geq 0}$). The range $\mathbb{Z}$ is used primarily to encode indexing functions for non-consecutive sets of states. The range $\mathbb{R}$ is used to encode the rates themselves.

Informally, decision diagrams are directed acyclic graphs organized in layers with each layer corresponding to a different variable in the domain of the function. The outgoing edges from a node correspond to the values the variable on that layer can take on. The

value of the function is determined by following the path from the root corresponding to the values taken by the domain variables. The path ends in a terminal node which, in BDDs and MDDs, give the value of the function.

A first proposal to encode non-binary function was to extend BDDs and MDDs so that, instead of the terminals 0 and 1, any element of X_0 can be a terminal node. The resulting *multi-terminal* (Clarke, Fujita, McGeer, Yang, & Zhao, 1993) BDDs (or MTMDDs) are quite general. However, as we will see, MTMDDs are sometimes unable to compactly encode even simple functions. We therefore focus on a newer class of *edge-valued* decision diagrams, which can be exponentially more compact, and provably never larger, than MTMDDs (Roux & Siminiceanu, 2010). For edge-valued decision diagrams, a value is associated with each edge in the tree, and the function's value is determined from the values along the path to the terminal node. The exact definition of these diagrams depends on the operator used to combine edge values. We consider two cases, EV$^+$MDDs (Ciardo & Siminiceanu, 2002) (where X_0 is either $\mathbb{Z} \cup \{\infty\}$ or $\mathbb{R} \cup \{\infty\}$ and edge values along a path are summed) and EV*MDDs (Wan, Ciardo, & Miner, 2011) (where X_0 is $\mathbb{R}^{\geq 0}$ and edge values along a path are multiplied).

4.2 Multiterminal and Edge-Valued Decision Diagrams

Formally, both an EV$^+$MDD and an EV*MDD are acyclic directed edge-labeled and edge-valued graphs. A node in the graph p has a level $p.lvl$ and a set of directed edges indexed by x. The edge associated with label x is written as $p[x] = \langle p[x].val, p[x].ch \rangle$, where $p[x].val$ is the value associated with the edge and $p[x].ch$ is the target of the edge.

- The only terminal node (one without outgoing edges) is Ω, at level 0: $\Omega.lvl = 0$.
- A nonterminal node p is at level $k \in \{1, \ldots, L\}$: $p.lvl = k$. For each $x_k \in X_k$, it has an outgoing edge labeled x_k, associated with a value $v \in X_0$, and pointing to a node q with $q.lvl < k$. Thus $p[x_k] = \langle v, q \rangle$.
- A node p at level k encodes the function $f_p : X_1 \times \cdots \times X_k \to X_0$. For EV$^+$MDDs, f_p is defined recursively as $f_p = 0$ if $p = \Omega$, and $f_p(x_1, \ldots, x_k) = p[x_k].val + f_{p[x_k].ch}(x_1, \ldots, x_{p[x_k].ch.lvl})$ otherwise (that is, if p is a nonterminal node).

  For EV*MDDs, f_p is defined recursively as $f_p = 1$ if $p = \Omega$, and $f_p(x_1, \ldots, x_k) = p[x_k].val \cdot f_{p[x_k].ch}(x_1, \ldots, x_{p[x_k].ch.lvl})$ otherwise.

Most decision diagram definitions have additional restrictions to ensure *canonicity*, that is so that any representable function has a unique representation. For the edge-valued decision diagrams we have defined, this is achieved by additionally requiring all of the following.

- There are no duplicate nodes: if $p.lvl = q.lvl = k$ and, for each $x_k \in X_k$, we have $p[x_k] = q[x_k]$, then $p = q$.
- The *absorbing value* terminates a path: for EV$^+$MDDs, $p[x_k].val = \infty$ implies that $p[x_k].ch = \Omega$; for EV*MDDs, $p[x_k].val = 0$ implies that $p[x_k].ch = \Omega$.
- Each node p at level $k > 0$ is *normalized*: for EV$^+$MDDs, $\min\{p[x_k].val : x_k \in X_k\} = 0$; for EV*MDDs, $\max\{p[x_k].val : x_k \in X_k\} = 1$.

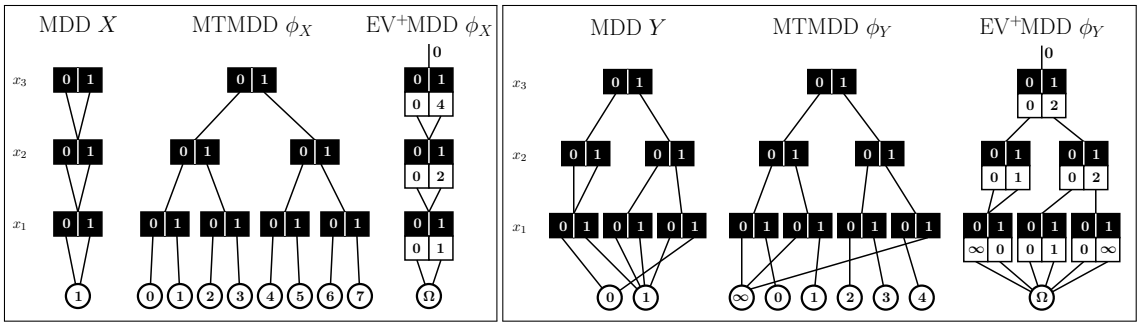


Figure 9: Encoding the lexicographic index function, ϕ_X, for set X. The left panel shows the quasi-reduced MDD encoding X followed by the MTMDD and the EV$^+$MDD encoding ϕ_X; the right panel shows the corresponding encodings for the set $Y = \{100, 110, 001, 101, 011\}$. In either case, the EV$^+$MDD is isomorphic to the MDD. Each level of the tree corresponds to a different variable. Black boxes are the values of this variable (and traversal of the diagram follows the edge leading out of this box for the value of input). White boxes (for EV$^+$MDD) are the values of the corresponding edge (which are summed to produce the function's value). The 0 at the top of the EV$^+$MDD is the value added to any path or traversal of the diagram.

Furthermore, we require that one of the following two *reduction forms* must be used.

- In *quasi-reduced* form, only nodes at level L have no incoming edges (except the special case of the graph consisting of just Ω) and the children of a node at level k are at level $k-1$ (except for absorbing-valued edges, which point to Ω, as stated above).

- In *fully-reduced* form, there are no redundant nodes, where a node p at level k is redundant if $p[x_k] = p[y_k]$ for all $x_k, y_k \in X_k$.

Strictly speaking, an EV$^+$MDD node encodes a function with values between 0 (included) and ∞ (possibly included); thus, a function f with range $\mathbb{Z} \cup \{\infty\}$ or $\mathbb{R} \cup \{\infty\}$ is encoded by $\langle \sigma, p \rangle$ where $\sigma = \min\{f(i) : i \in X\}$ and $f_p = f - \sigma$ (the special case $f \equiv \infty$ is encoded by the pair $\langle \infty, \Omega \rangle$). Analogously, an EV*MDD node encodes a function with values between 0 (possibly included) and 1 (included); thus a function f with range $\mathbb{R}^{\geq 0}$ is encoded by $\langle \sigma, p \rangle$ where $\sigma = \max\{f(i) : i \in X\}$ and $f_p = f/\sigma$ (the special case $f \equiv 0$ is encoded by the pair $\langle 0, \Omega \rangle$). In the following, we use the term EV$^+$MDD or EV*MDD also for the pair $\langle \sigma, p \rangle$, with the understanding that σ is just a parameter that scales the values of the function encoded by node p.

4.3 Lexicographic Index Example

We illustrate the compactness of these decision diagrams using the *lexicographic index*, also called the *mixed-base value*, of a state $x = (x_1, \ldots, x_L)$, defined as $\phi(x) = \sum_{1 \leq k \leq L} x_k \cdot n_{1:k-1}$,

where $n_{a:b} = n_a \cdots n_b$ for $a \leq b$ (as in Section 3.1). We discuss the importance of this function after showing its encoding.

Figure 9 (left) shows the lexicographic index function ϕ (along side the MDD encoding the set of states). The MTMDD for ϕ is a full L-level tree with $n_{1:L}$ leaves. By contrast, the EV$^+$MDD for ϕ contains just one node at each level, where the child labeled x_k of the node at level k points to the node at level $k-1$ and has value $x_k \cdot n_{1:k-1}$.

Interestingly, this function retains a compact encoding even if we modify it so that it applies to a set $Y \subset X$, that is $\phi_Y(x) = |\{y \in Y : \phi(y) < \phi(x)\}|$ if $x \in Y$, and $\phi_Y(x) = \infty$ otherwise, in the sense that the MDD encoding Y and the EV$^+$MDD encoding ϕ_Y are isomorphic (right panel in Figure 9). This is of particular importance for the exact numerical solution of structured CTMPs whose reachable state space X_{rch} is a strict (and possibly complicated) subset of X, since in this case we need to frequently and efficiently map a state $x = (x_1, \ldots, x_L)$ to its index $\phi_{X_{rch}}(x)$ in a full probability vector of size $|X_{rch}|$. Compactly representing this index function is key to efficient calculations in such CTMPs.

Obviously, EV*MDDs can also be exponentially more compact than MTMDDs; simply consider that the EV*MDD encoding of $e^{-\phi}$ also has one node per level, where the child with label x_k of the node at level k has value $e^{-x_k \cdot n_{1:k-1}}$.

4.4 Decision Diagram Operations

In addition to efficiently *encoding* structured functions, decision diagrams are also able to efficiently *manipulate* functions. All decision diagram operations proceed recursively from the root node(s) and make extensive use of dynamic programming. Specifically, they use an *operation cache* to retrieve the result of a specific operation on a specific choice of parameters, if this result has been previously computed when exploring different paths in the recursion. This reduces the worst-case complexity of an operation (for example, computing $c = a + b$, where a and b are functions encoded by two EV*MDDs) from exponential (i.e., the size of the domain) to polynomial. For example, Figure 10 shows the pseudocode for an algorithm to perform the element-wise addition of two EV*MDDs, that is when $\alpha, \beta \geq 0$ and a, b are EV*MDD nodes at level L (unless $\alpha = 0$, in which case $a = \Omega$, or $\beta = 0$, in which case $b = \Omega$), Sum$(L, \langle\alpha,a\rangle, \langle\beta,b\rangle)$ returns an EV*MDD $\langle\rho,r\rangle$ such that, for all $x = (x_1, \ldots, x_L) \in X$, we have $\rho \cdot f_r(x) = \alpha \cdot f_a(x) + \beta \cdot f_b(x)$; of course, the input EV*MDDs are assumed to be in canonical form, and the output EV*MDD is guaranteed to be in the same canonical form. Its complexity is the product of the sizes of the input EV*MDDs.

In a practical implementation, the "unique table," which stores nodes and avoids duplicates, is implemented as a lossless hash table which, given a lookup key $\langle level, r[0], \ldots, r[n_k - 1]\rangle$, returns a node's address, while the cache is implemented as a (possibly) lossy hash table. The cache can be made more effective by scaling and exploiting commutativity. For example, by defining an arbitrary order on nodes (for example, $a \prec b$ if the memory address of a is smaller than that of b), we can exchange the two input EV*MDDs to ensure that $a \prec b$ prior to cache lookup, and then observe that $\alpha \cdot f_a + \beta \cdot f_b = \alpha \cdot (f_a + \gamma \cdot f_b)$, for $\gamma = \beta/\alpha$, so that we just store entries of the form $\langle SUM, a, \gamma, b \to \rho, r\rangle$ in the cache. Then, assuming $p \prec q$, the call Sum$(k, \langle 0.5,p\rangle, \langle 0.2,q\rangle)$ would be cached as $\langle SUM, p, 0.4, q \to \sigma, s, \rangle$ and

function Sum(level k, EV*MDD$\langle\alpha,a\rangle$, EV*MDD$\langle\beta,b\rangle$)
  if $a = b$ **then return** $\langle\alpha+\beta,a\rangle$ ▷ This includes the terminal case $k = 0$: $a = b = \Omega$
  if $\alpha = 0$ **then return** $\langle\beta,b\rangle$ ▷ $a = \Omega$ by definition
  if $\beta = 0$ **then return** $\langle\alpha,a\rangle$ ▷ $b = \Omega$ by definition
  if *cache* contains $\langle SUM, \alpha, a, \beta, b \to \rho, r\rangle$ **then return** $\langle\rho,r\rangle$ ▷ Check if result is in the cache
  $r \leftarrow$ NewNode(k) ▷ Create new temporary result node at level k
  for all $x_k \in X_k$ **do**
    $r[x_k] \leftarrow$ Sum($k-1$,$\langle\alpha \cdot a[x_k].val, a[x_k].ch\rangle$,$\langle\beta \cdot b[x_k].val, b[x_k].ch\rangle$) ▷ Recurse down one level
  $\rho \leftarrow \max_{x_k \in X_k}\{r[x_k].val\}$; ▷ Maximum edge value for node r before normalization
  for all $x_k \in x_k$ **do**
    $r[x_k].val \leftarrow r[x_k].val/\rho$ ▷ Normalize node r so that the maximum edge value is 1
  $r \leftarrow$ UniqueTableInsert(r); ▷ If node like r exists, return it and delete r, else return r
  Enter $\langle SUM, \alpha, a, \beta, b \to \rho, r\rangle$ in *cache*; ▷ Remember the result in the operation cache
  return $\langle\rho,r\rangle$;

Figure 10: Pseudo-code for sum of quasi-reduced EV*MDDs (a and b are either Ω or nodes at level k). The fully-reduced version is similar but slightly more involved, as it needs to take into account the levels of a and b.

return $\langle 0.5\sigma,s\rangle$, while a subsequent call Sum$(k, \langle 0.25,p\rangle, \langle 0.1,q\rangle)$ or Sum$(k, \langle 0.1,q\rangle, \langle 0.25,p\rangle)$ would find $\langle SUM, p, 0.4, q \to \sigma, s, \rangle$ in the cache and immediately return $\langle 0.25\sigma,s\rangle$.

4.5 Encoding Transition Rate Matrices with EV*MDDs

We now turn to the use of EV*MDDs to compactly encode the transition rate matrix $\boldsymbol{R}$ (the same as the intensity matrix $\boldsymbol{Q}$, but without the diagonal) of a CTMP. This can be accomplished using various approaches.

4.5.1 Monolithic Encoding vs. Disjunctive Partition Encoding

Clearly, a node r of a $2L$-level EV*MDD can encode an arbitrary function of the form $X \times X \to [0,1]$. Then, for $\sigma \geq 0$, the pair $\langle\sigma,r\rangle$ encodes an arbitrary function of the form $X \times X \to [0,\sigma]$, where EV*MDD levels $(1,\ldots,2L)$ correspond to state variables $(x'_1, x_1, \ldots, x'_L, x_L)$, that is, we use an interleaved order to describe the transition rate from x to x'. With a *monolithic* approach, we can then store $\boldsymbol{R}$ using a single EV*MDD $\langle\sigma,r\rangle$ where σ is the largest rate in $\boldsymbol{R}$ and r encodes matrix $\boldsymbol{R}/\sigma$.

However, many practical systems exhibit *asynchronous* behavior, that is each state change is due to some *event* $e \in E$ occurring (asynchronously) somewhere in the system. In these situations, we can employ a *disjunctive partition* to encode $\boldsymbol{R}$, storing a set of EV*MDDs $\{\langle\sigma_e,r_e\rangle : e \in E\}$, so that $\langle\sigma_e,r_e\rangle$ encodes matrix $\boldsymbol{R_e}$, where $R_e(x,x')$ describes the rate at which the system moves from state x to state x' due to the occurrence of event e.

With this disjunctive partition encoding, $\boldsymbol{R} = \sum_{e \in E} \boldsymbol{R_e}$ does not have to be built explicitly; rather, the individual matrices $\boldsymbol{R_e}$ are directly used in the numerical computations used to solve the CTMP. The idea of a disjunctive partition was initially suggested for BDDs (Burch, Clarke, & Long, 1991), although it is obviously also related to Kronecker encodings: consider Equation 76, which expresses $\boldsymbol{R}$ first and foremost as a sum.

The choice between a monolithic or a disjunctive partition encoding is largely model-dependent. In most applications, the high-level language description of the model suggests what the set of asynchronous events E should be. Thus, we first build the EV*MDDs for the disjuncts $\boldsymbol{R_e}$ then, if desired, we can explicitly build the EV*MDD for $\boldsymbol{R}$ by summing the EV*MDDs for the disjunct corresponding to each event (using the algorithm in Figure 10, for instance).

However, while the disjunct EV*MDDs are usually quite compact, the EV*MDD for $\boldsymbol{R}$ obtained by summing the disjuncts $\boldsymbol{R_e}$ might still be very compact, or it might grow very large. In the former case, the monolithic approach is preferable, as it allows us to directly use the EV*MDD encoding $\boldsymbol{R}$ in the numerical iterations. In the latter case, the disjunctive partition approach is preferable, as it allows us to use the EV*MDD for each $\boldsymbol{R_e}$ individually, without even attempting to build the monolithic EV*MDD encoding $\boldsymbol{R}$.

For example, consider a simple system of four Boolean variables, X_1, X_2, X_3, and X_4, and two events. The first event, c_{21}, changes the value of (X_2, X_1), interpreted as a 2-bit integer, in the sequence $0 \text{–}[1]\!\rightarrow 1 \text{–}[1/2]\!\rightarrow 2 \text{–}[1/4]\!\rightarrow 3 \text{–}[1/8]\!\rightarrow 0 \text{–}[1]\!\rightarrow \cdots$, where the numbers in the square brackets indicate the rate of the corresponding transition. The second event, c_{43}, changes the value of (X_4, X_3), interpreted as a 2-bit integer, in the sequence $0 \text{–}[3]\!\rightarrow 1 \text{–}[1]\!\rightarrow 2 \text{–}[1/3]\!\rightarrow 3 \text{–}[1/9]\!\rightarrow 0 \text{–}[3]\!\rightarrow \cdots$. Figure 11 on the left shows the EV*MDDs encoding the matrices $\boldsymbol{R_{21}}$ and $\boldsymbol{R_{43}}$ corresponding to these two events, as well as the matrix $\boldsymbol{R} = \boldsymbol{R_{21}} + \boldsymbol{R_{43}}$. (for visual simplicity, edges with value 0, which by definition point to the terminal node Ω, are not shown).

4.5.2 Adopting Ideas from Kronecker Encodings: Identity Patterns

Neither the monolithic approach nor the disjunctive partition approach exploit *locality*: the fact that most events (synchronously) affect only a few state variables. In other words, while each matrix $\boldsymbol{R_e}$ is conceptually of size $|X| \times |X|$, it usually has a much smaller *support* $S_e \subseteq \{x_1, \ldots, x_L\}$. Specifically, $X_k \in S_e$ if and only if X_k and e are *dependent*: the local state x_k affects the rate at which e occurs (including the case where it may disable e altogether, that is set its rate to 0) or is changed by the occurrence of e. When $X_k \notin S_e$, X_k and e are *independent* and the EV*MDD encoding $\boldsymbol{R_e}$ contains *identity patterns* in correspondence to x_k. For example, the EV*MDD encoding $\boldsymbol{R_{21}}$ in Figure 11 on the left exhibits such patterns with respect to variables X_4 and X_3, while the one for $\boldsymbol{R_{43}}$ exhibits them for X_2 and X_1.

Essentially, these identity patterns simply describe the fact that the value of x'_k (the new value of x_k after the occurrence of e) equals the old value of X_k and that the rate is not affected by the value of X_k, and this is true for all possible values of X_k. When this happens, the Kronecker encoding of event e has $\boldsymbol{R_e^{(k)}} = \boldsymbol{I_{n_k}}$, while the quasi-reduced or the fully-reduced forms alone cannot take advantage of these common patterns. To exploit these patterns, a combination of the fully-reduced form, for unprimed level X_k, and a new

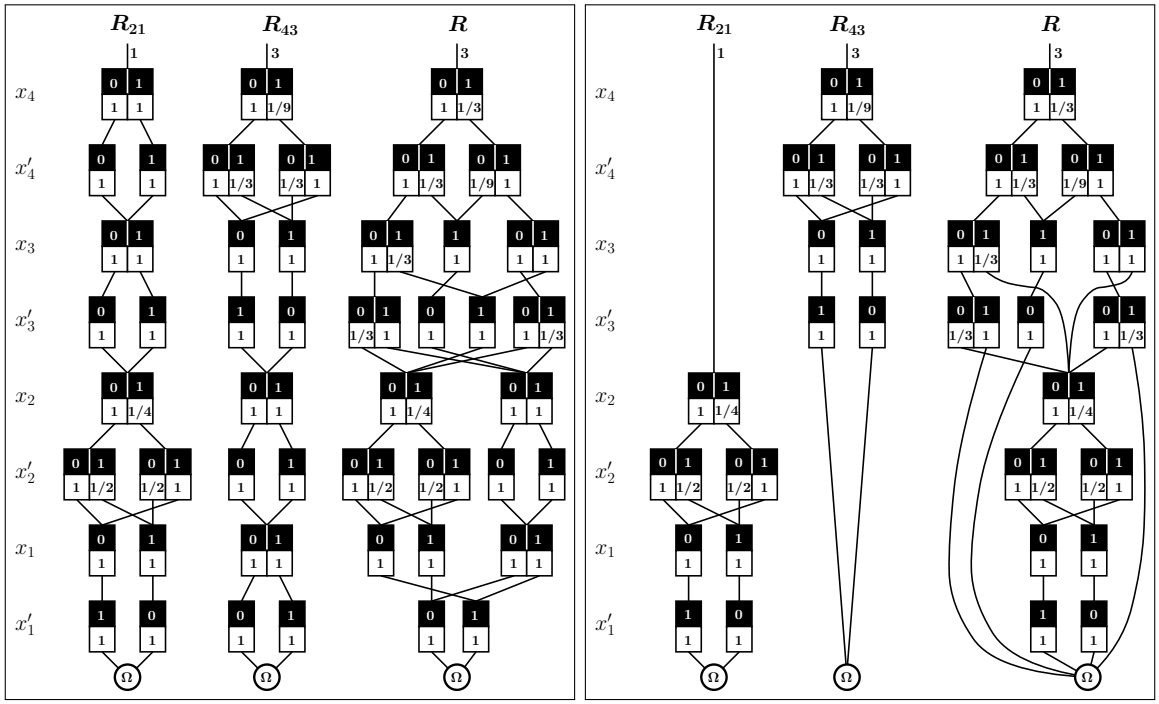


Figure 11: An example of EV*MDDs encoding transition rate matrices using the quasi-reduced form (left) or the fully-identity-reduced form (right). Omitted edges have implied value 0 (thus resulting in value 0 for any path containing them). The right diagrams are the same as those on the left, except that identity patterns have been omitted and are implied: Any completely skipped pair of levels is assumed to have an identity structure (compare to corresponding diagram on the left).

identity-reduced form (Ciardo & Yu, 2005) for primed level X'_k, is needed. This allows us to encode $\boldsymbol{R_e}$ in an EV*MDD which has nodes only at (unprimed and primed) levels corresponding to state variables in S_e. A further advantage of this *fully-identity-reduced* form is that the resulting decision diagrams, unlike a Kronecker encoding, also recognize and exploit *partial* identity patterns (those arising in models where X_k remains unchanged after e occurs in certain states but not in others). Figure 11 on the right shows the encoding of the same matrices $\boldsymbol{R_{21}}$, $\boldsymbol{R_{43}}$, and $\boldsymbol{R}$, but using this new fully-identity-reduced form.

4.5.3 Beyond Kronecker: Disjunctive-then-Conjunctive Partition Encoding

We can push the decomposition further by employing a *disjunctive-then-conjunctive partition* approach. This idea was first introduced for logic analysis (Ciardo & Yu, 2005) but it is also related to Equation 76, which expresses $\boldsymbol{R}$ as a sum of products. This is particularly appropriate for *globally-asynchronous locally-synchronous* systems, where not only each state change is due to an (asynchronous) event $e \in E$, but the occurrence of e de-

pends on and (synchronously) changes only a few state variables. Each $\boldsymbol{R_e}$ is then further decomposed into the product of m matrices, $\boldsymbol{R_e} = \prod_{1 \leq c \leq m} \boldsymbol{R_e^{(c)}}$ and, again, each matrix $\boldsymbol{R_e^{(c)}}$ is conceptually of size $|X| \times |X|$ but, in practice, it usually has a small support $S_e^{(c)}$. Specifically, $X_k \in S_e^{(c)}$ if and only if the fully-identity-reduced EV*MDD for $\boldsymbol{R_e^{(c)}}$ contains a node associated to X_k or X'_k. We restrict ourselves to the case where the supports of the conjuncts for an event are disjoint, so that $\bigcup_{1 \leq c \leq m} S_e^{(c)} = S_e$ and each $S_e^{(c)}$ is substantially smaller than S_e. For example, when a Kronecker encoding for $\boldsymbol{R_e}$ exists, that is $\boldsymbol{R_e} = \bigotimes_{1 \leq k \leq L} \boldsymbol{R_e^{(k)}}$, we have $S_e^{(k)} = \{X_k\}$ for each $X_k \in S_e$, that is for each $\boldsymbol{R_e^{(k)}} \neq \boldsymbol{I_{n_k}}$; in this case, a disjunctive-then-conjunctive approach that uses an EV*MDD to store each $\boldsymbol{R_e^{(c)}}$ is as compact as the disjunctive partition approach that uses an EV*MDD to store each $\boldsymbol{R_e}$, and both are essentially just as compact as the Kronecker approach (except that they can save additional memory by exploiting partial identity patterns).

The disjunctive-then-conjunctive approach is instead distinctly more efficient when the Kronecker approach is not applicable (that is, when the Kronecker approach would require an enormous set of events to correctly describe $\boldsymbol{R}$), but it can nevertheless be seen as an extension of the Kronecker approach. Consider using the decomposition of Equation 75:

$$\begin{aligned}\boldsymbol{R_e} = \bigotimes_{1 \leq k \leq L} \boldsymbol{R_e^{(k)}} &= \prod_{1 \leq k \leq L} \boldsymbol{P_{n_{1:k}, n_{k+1:L}}}^{\top} \cdot (\boldsymbol{I_{\overline{n}_k}} \otimes \boldsymbol{R_e^{(k)}}) \cdot \boldsymbol{P_{n_{1:k}, n_{k+1:L}}} \\ &= \prod_{X_k \in S_e} \boldsymbol{P_{n_{1:k}, n_{k+1:L}}}^{\top} \cdot (\boldsymbol{I_{\overline{n}_k}} \otimes \boldsymbol{R_e^{(k)}}) \cdot \boldsymbol{P_{n_{1:k}, n_{k+1:L}}}\end{aligned}$$

where the last step simply stresses that, when $\boldsymbol{R_e^{(k)}} = \boldsymbol{I_{n_k}}$, the corresponding factor is just $\boldsymbol{I_{n_{1:L}}}$ and can be skipped.

This is the idea behind the *Shuffle Algorithm* (Fernandes, Plateau, & Stewart, 1998), which, as observed by Buchholz, Ciardo, Donatelli, and Kemper (2000), is very efficient, but only when the matrices $\boldsymbol{R_e^{(k)}}$ are not "too sparse." (The perfect shuffle pre- and post-multiplications are essentially free; they simply describe a different state indexing.)

Then, the disjunctive-then-conjunctive approach extends the Kronecker expression of $\boldsymbol{R_e}$ to allow situations where the factors are not restricted to a support consisting of just one variable, but still exploits each factor's locality:

$$\boldsymbol{R_e} = \prod_{1 \leq c \leq m} \boldsymbol{P_{S_e^{(c)}}}^{\top} \cdot \left(\boldsymbol{I_{\overline{S}_e^{(c)}}} \otimes \boldsymbol{R_{S_e^{(c)}}} \right) \cdot \boldsymbol{P_{S_e^{(c)}}} \tag{77}$$

where $\boldsymbol{P_{S_e^{(c)}}}^{\top}$ and $\boldsymbol{P_{S_e^{(c)}}}$ are perfect shuffle permutations that respectively move all dependent state variables in $S_e^{(c)}$ at the end of the variable order and back to their original position, $\boldsymbol{I_{\overline{S}_e^{(c)}}}$ is the identity matrix of size $\prod_{X_h \notin S_e^{(c)}} n_h$ (the skipped levels), and $\boldsymbol{R_{S_e^{(c)}}}$ is a square matrix of size $\prod_{X_h \in S_e^{(c)}} n_h$ (the conjunct encoded by an EV*MDD if we ignore the skipped levels corresponding to state variables not in $S_e^{(c)}$).

Since the supports $S_e^{(c)}$ are disjoint, this generalization of the Kronecker approach comes at no additional cost and essentially corresponds to a Kronecker approach where we allow each event to be defined on a different set of state variables, each set corresponding to

a different partition of the "basic" state variables $(X_1, .., X_L)$. In this case, building the EV*MDD for $\boldsymbol{R_e}$ by multiplying the EV*MDDs for each $\boldsymbol{R_{S_e^{(c)}}}$ does not involve any numeric multiplication, but it grows the size of the diagram if the spans of the sets $S_e^{(c)}$ are not disjoint; for example, if $S_e^{(1)} = \{X_3, X_7\}$ and $S_e^{(2)} = \{X_4, X_6\}$, then each path from X_7 to X_3 in the EV*MDD for $\boldsymbol{R_e}$ will contain a copy of the entire EV*MDD encoding $\boldsymbol{R_{e^{(2)}}}$.

4.5.4 Numerical Solutions with Decision Diagrams

We have described a method for storing the rate or intensity matrix compactly for common process models. We now address the use of these data structures in CTMP computations. The literature surrounding decision diagrams and CTMPs is primarily concerned with computation of the unconditional distribution of the resulting process, either at a finite time or (more commonly) in the limit of infinite time (the stationary distribution). We follow the convention of this literature and refer to this as the *solution* of the process. Model estimation, solutions conditioned on evidence, and computations of marginals or other statistics are, to our knowledge, unexplored for these representations, but we return to this later.

When matrix $\boldsymbol{R}$ is stored using $2L$-level EV*MDDs (using a monolithic, a disjunctive partition, or a disjunctive-then-conjunctive partition encoding), the traditional numerical solution algorithms need to be adjusted accordingly. First of all, when seeking an *exact* solution, neither the stationary vector $\boldsymbol{\pi}$ nor the transient vector $\boldsymbol{\pi}(t)$ admit a compact EV*MDD representation (unless the modeled system contains extensive symmetries or is composed of completely independent subsystems, which is rarely the case in practice).

Two approaches have been explored. For an exact solution for the stationary distribution $\boldsymbol{\pi}$ (the null-space of $\boldsymbol{Q}$), a *hybrid* approach (Kwiatkowska, Norman, & Parker, 2004) is usually best, where the solution is stored as a full vector of reals having size equal to the number of reachable states ($|X_{rch}|$, equal to $|X|$ only if all states are reachable) while the rate matrix $\boldsymbol{R}$ is stored with EV*MDDs, and the *expected holding time* vector h (the inverse absolute values of the diagonal of $\boldsymbol{Q}$) is stored either as a full vector or with EV*MDDs. Clearly, such an approach scales the size of the tractable problems by eliminating the main memory obstacle (the storage of $\boldsymbol{R}$), only to encounter the next memory obstacle (the storage of the solution vector). For example, Figure 12 shows the pseudocode for a classic Jacobi-style stationary solution of an ergodic CTMP when the transition rate matrix is monolithically encoded by the EV*MDD $\langle\sigma,r\rangle$, while the state space X_{rch} is indexed by an EV$^+$MDD $\langle 0,p\rangle$, as previously discussed, so that, for each $i \in X$, we can compute $\phi_{X_{rch}}(i)$, an index between 0 and $|X_{rch}| - 1$ included if $i \in X_{rch}$, or ∞ if $i \notin X_{rch}$. Function $\phi_{X_{rch}}$ is used to index entries of the solution vector: $\boldsymbol{\pi^{new}}$ and $\boldsymbol{\pi^{old}}$. The holding time vector is stored as the full vector h, also indexed by $\phi_{X_{rch}}$ (but it could have been stored using EV*MDDs instead). At each recursive call of JacobiRecur, we descend a "from" and a "to" level from the current rate matrix EV*MDD node and a single level from the corresponding "source" and "destination" EV$^+$MDD nodes (these are needed to index the full vectors of reals, and are initially both set to $\langle 0,p\rangle$, encoding the entire $\phi_{X_{rch}}$ function). Note that, in the simple case when all states are reachable (that is, $X = X_{rch}$) the indexing function $\phi_{X_{rch}}$ is just the mixed-base value $\phi(i) = \sum_{1\le k\le L} i_k \cdot n_{1:k-1}$ discussed in Section 4.3 and, as such, it does not really require an EV$^+$MDD for its encoding; on the other hand, as shown in Figure 9, this EV$^+$MDD is just a single path of nodes, so its use does not carry

```
▷ Computes π such that πQ = 0. ⟨σ,r⟩ is R, ⟨0,p⟩ is φ_{X_rch}
function JacobiIteration(EV*MDD⟨σ,r⟩,EV+MDD⟨0,p⟩)
    π^old ← "initial guess"          ▷ Real vector of size |X_rch|, visible to JacobiRecur
    num_iter ← 0
    repeat
        π^new ← zero vector          ▷ Real vector of size |X_rch|, visible to JacobiRecur
        JacobiRecur(L,⟨σ,r⟩,⟨0,p⟩,⟨0,p⟩)
        for all i ∈ {0,...,|X_rch| − 1} do
            π^new[i] ← π^new[i] · h[i]          ▷ h is the holding time vector
        swap(π^old, π^new)
        num_iter ← num_iter + 1
    until num_iter > MAX_ITER or Converged(π^old,π^new)
    ▷ for example, using a relative or absolute test

▷ Computes π^new ← π^old R
function JacobiRecur(level k, EV*MDD⟨σ,m⟩, EV+MDD⟨η_src,src⟩,
EV+MDD⟨η_des,des⟩)
    if src = des = Ω then
        π^new[η_des] ← π^new[η_des] + π^old[η_src] · σ
        return
    for i from 0 to n_k−1 s.t. m[i].val ≠ 0 and src[i].val ≠ ∞ do      ▷ "from" level k
        for j from 0 to n_k−1 s.t. m[i][j].val ≠ 0 and des[j].val ≠ ∞ do   ▷ "to" level k'
            η'_src ← η_src + src[i].val
            η'_des ← η_des + des[j].val
            σ' ← σ · m[i][j].val
            JacobiRecur(k−1, ⟨σ',m[i][j].ch⟩, ⟨η'_src,src[i].ch⟩, ⟨η'_des,des[j].ch⟩)
```

Figure 12: A Jacobi-style iteration for the stationary solution ($\boldsymbol{\pi Q} = \frac{d\boldsymbol{\pi}}{dt} = \mathbf{0}$) when $\boldsymbol{R}$ (non-diagonal elements of $\boldsymbol{Q}$) is stored as a monolithic EV*MDD and $\boldsymbol{h}$ (inverse absolute value of the diagonal elements of $\boldsymbol{Q}$) is stored in a vector. $\langle\sigma,r\rangle$ is the encoding of $\boldsymbol{R}$ and $\langle 0,p\rangle$ is the encoding of the mapping from states to indices (for $\boldsymbol{\pi}$ and $\boldsymbol{h}$).

any overhead. A similar hybrid approach can be used to compute a transient solution using a uniformization-style algorithm where $\boldsymbol{R}$ is also stored using EV*MDDs but, again, the size of the full vectors limits scalability.

The filtering and smoothing operations as described in Section 2.5 have not been explicitly tackled for decision-diagram encodings. However, if we are willing to represent the distribution exactly (as above), we can do the necessary vector-matrix multiplications directly on the decision-diagrams (without expanding them). Estimating an EV*MDD representation of $\boldsymbol{R}$ from data is completely unexplored.

To tackle larger problems we must instead be willing to accept an approximate solution. However, work in this area is mostly restricted to systems exhibiting special structures.

One exception is the work of Wan et al. (2011), which addresses the stationary solution of arbitrary ergodic CTMPs whose state space is encoded as an MDD and whose transition rate matrix is encoded as one or more EV*MDDs. The approach uses L different approximate aggregations of the exact CTMP and solves them iteratively, until reaching a fixpoint. This approach provides an exact solution under certain conditions — essentially, if the system has a so-called "product-form" (Baskett, Chandy, Muntz, & Palacios-Gomez, 1975). Unfortunately, no similar approximation for the transient solution of a structured CTMP has been proposed so far.

Thus for state spaces large enough that a single vector over their values cannot be maintained, the literature for inference and estimation with these models is very limited. However, in the next section we describe a different model that can be viewed as a restricted form of the disjunctive EV*MDD encoding of this section. This model has many inference and estimation method and we believe this link between the two may allow for those methods to be extended to more general decision-diagram representations.

5. Continuous-Time Bayesian Networks

In the artificial intelligence and machine learning literatures, continuous-time Bayesian networks (CTBNs) (Nodelman, Shelton, & Koller, 2002) were developed as an extension of dynamic Bayesian networks (DBNs). As we discuss in this section, they are a more limited case of the disjunctive EV*MDD encodings above. However, approximate methods and computations conditioned on evidence have been more extensively developed for CTBNs.

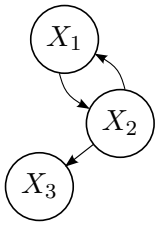
A CTBN consists of a set of variables, $\{X_1, X_2, \ldots, X_L\}$, a directed graph $\mathcal{G}$ with a one-to-one mapping between the nodes and the variables, a set of conditional intensity matrices for each variable, and an initial distribution. The initial distribution is usually described as a Bayesian network (to keep its description compact), but many of the algorithms and theory hold for other compact distribution representations.

The graph $\mathcal{G}$ describes instantaneous influence of variables on each other. An edge from X_i to X_j denotes that the rates of transitions of X_j depend on the instantaneous value of X_i. Note that $\mathcal{G}$ may be cyclic.

These dependent rates are captured in the conditional intensity matrices. Let n_i be the number of states for variable X_i. We denote the parents of variable X_i as Par_i and a joint assignment to Par_i as $\mathbf{par_i}$. The set of conditional intensity matrices for variable X_i consists of one n_i-by-n_i intensity matrix for each possible instantiation $\mathbf{par_i}$: $\boldsymbol{Q}_{\boldsymbol{X_i}|\mathbf{par_i}}$ for which we denote element x_i, x_i' as $q_{x_i,x_i'|\mathbf{par_i}}$, the rate of X_i transitioning from x_i to x_i' when Par_i have values $\mathbf{par_i}$.

Semantically, a CTBN is a continuous-time Markov process over the joint state space of all constituent variables. We let $\delta(\boldsymbol{x}, \boldsymbol{x'})$ be equal to the set of variables whose assignments differ between joint assignments $\boldsymbol{x}$ and $\boldsymbol{x'}$. The joint intensity matrix for the entire process can be described as

$$q_{\boldsymbol{x},\boldsymbol{x'}} = \begin{cases} \sum_{i=1}^{L} q_{x_i,x_i'|\mathbf{par_i}} & \text{if } \delta(\boldsymbol{x}, \boldsymbol{x'}) = \{\} \\ q_{x_i,x_i'|\mathbf{par_i}} & \text{if } \delta(\boldsymbol{x}, \boldsymbol{x'}) = \{X_i\} \\ 0 & \text{otherwise} \end{cases} \tag{78}$$



$$\boldsymbol{Q}_{\boldsymbol{X_1|X_2=0}} = \begin{bmatrix} -1 & 1 \\ 2 & -2 \end{bmatrix} \qquad \boldsymbol{Q}_{\boldsymbol{X_1|X_2=1}} = \begin{bmatrix} -4 & 4 \\ 3 & -3 \end{bmatrix}$$

$$\boldsymbol{Q}_{\boldsymbol{X_2|X_1=0}} = \begin{bmatrix} -5 & 5 \\ 6 & -6 \end{bmatrix} \qquad \boldsymbol{Q}_{\boldsymbol{X_2|X_1=1}} = \begin{bmatrix} -7 & 7 \\ 8 & -8 \end{bmatrix}$$

$$\boldsymbol{Q}_{\boldsymbol{X_3|X_2=0}} = \begin{bmatrix} -9 & 0 & 9 \\ 8 & -10 & 2 \\ 7 & 4 & -11 \end{bmatrix} \qquad \boldsymbol{Q}_{\boldsymbol{X_3|X_2=1}} = \begin{bmatrix} -12 & 6 & 6 \\ 8 & -13 & 5 \\ 7 & 7 & -14 \end{bmatrix}$$

Figure 13: An example CTBN graph. See Figures 14, 15, and 17 for the same CTMP in other representations.

where $\mathbf{par}_{\boldsymbol{i}}$ is the assignment to Par_i in $\boldsymbol{x}$. The intensity of transition between two states that differ only by one variable can be read from the appropriate conditional intensity matrix for that variable. The intensity of transition between any other two states (that differ by more than one variable) is zero. The diagonal elements are filled in to be the negative row sums. This process allows two variables to transition at arbitrarily close times, but not at exactly the same time.

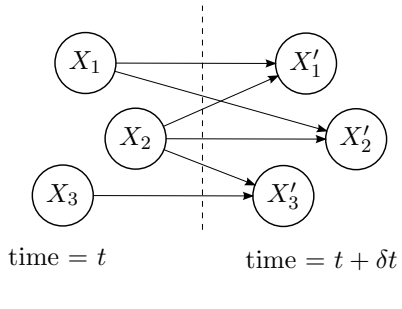
A CTBN retains the local Markov properties of a standard Bayesian network. In particular, a variable (local process) is independent of its non-descendants, given its parents. Of course, because of cycles, parents may also be descendants, but this does not pose a problem to the definition. Note, however, that *given* refers to conditioning on the entire trajectory of a variable (from the starting time until the ending time, after which no variables are queried or observed). Conditioning on the current value is not sufficient (even for rendering only the current values independent).

The global Markov properties also hold. The Markov blanket for a variable is the union of the sets of its parents, its children, and its children's parents. Note that these sets can have significant overlap, as cycles are permitted. Conditioned on its Markov blanket, a variable is independent of all other variables.

5.1 Connections to Other Representations

A CTBN can be related to a number of other representations. For instance, Portinale and Codetta-Raiteri (2009) link CTBNs to stochastic Petri nets (Ajmone Marsan, Balbo, Conte, Donatelli, & Franceschinis, 1995). Donatelli (1994) shows the translation from stochastic Petri nets to Kronecker operators and Ciardo, Zhao, and Jin (2012) show the translation from (ordinary, timed, or stochastic) Petri nets to various classes of decision diagrams. However, below we concentrate on more direct comparisons of the approaches presented in this tutorial.

Figure 13 shows a simple small CTBN of two binary variables (X_1 and X_2) and one ternary variable (X_3). We will use this as a running example for how to convert from a CTBN to other compact representations.



$P(X_1'|X_1, X_2)$:

	$0,0$	$1,0$	$0,1$	$1,1$
0	$1-1\delta t$	$2\delta t$	$1-4\delta t$	$3\delta t$
1	$1\delta t$	$1-2\delta t$	$4\delta t$	$1-3\delta t$

$P(X_2'|X_2, X_1)$:

	$0,0$	$1,0$	$0,1$	$1,1$
0	$1-5\delta t$	$6\delta t$	$1-7\delta t$	$8\delta t$
1	$5\delta t$	$1-6\delta t$	$7\delta t$	$1-8\delta t$

$P(X_3'|X_3, X_2)$:

	$0,0$	$1,0$	$2,0$	$0,1$	$1,1$	$2,1$
0	$1-9\delta t$	$8\delta t$	$7\delta t$	$1-12\delta t$	$8\delta t$	$7\delta t$
1	0	$1-10\delta t$	$4\delta t$	$6\delta t$	$1-13\delta t$	$7\delta t$
2	$9\delta t$	$2\delta t$	$1-11\delta t$	$6\delta t$	$5\delta t$	$1-14\delta t$

Figure 14: A DBN whose limit as $\delta t \to 0$ approaches the CTBN of Figure 13.

5.1.1 Connection to DBN

From a CTBN, we can construct a dynamic Bayesian network (DBN) whose parameters are a function of the time between slices such that its limit as this time-slice width approaches zero is the original CTBN. In particular, the DBN has no intra-time-slice edges (this is because two variables in the CTBN cannot change at exactly the same time). If X_i has parents Par_i in the CTBN, then it has the same parents (at the previous time slice) plus its previous value in the DBN. Figure 14 shows the CTBN of Figure 13 as a DBN. If X_i in the CTBN has an intensity matrix $\boldsymbol{Q}_{\boldsymbol{X_i}|\mathbf{par}_{\boldsymbol{i}}}$ for parent values $\mathbf{par}_{\boldsymbol{i}}$, then the corresponding variable in the DBN has the conditional probability distribution

$$p_{\text{DBN}}(x_i'|x_i, \mathbf{par}_{\boldsymbol{i}}) = \delta_{x_i', x_i} + \delta t \, q_{x_i, x_i'|\mathbf{par}_{\boldsymbol{i}}} \tag{79}$$

where x_i' is the value of X_i at the "next" time step (and x_i and $\mathbf{par}_{\boldsymbol{i}}$ are the values at the "previous" time step), δ_{x_i', x_i} is 1 if $x_i' = x_i$ and 0 otherwise, and δt is the time between time slices. The limit of this process as δt approaches 0 is the original CTBN.

5.1.2 Connection to Kronecker Algebra

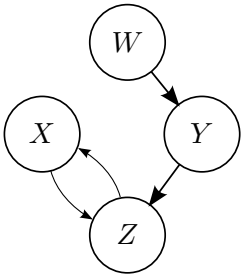
The joint intensity matrix expressed in Equation 78 can also be written as a sum of Kronecker products. We need first to define a conceptually simple, but notationally cumbersome, term. First, let $\boldsymbol{\Delta}_{\boldsymbol{i,j}}$ be a matrix of all 0, except for a single 1 in location i, j (same as before). Second, let $\tilde{\boldsymbol{Q}}_{\boldsymbol{X_i}|\mathbf{par}_{\boldsymbol{i}}}$ denote the Kronecker product of one matrix for each variable in the CTBN. If the variable is X_i, the matrix is $\boldsymbol{Q}_{\boldsymbol{X_i}|\mathbf{par}_{\boldsymbol{i}}}$. If the variable is a parent of X_i and has value x_k in $\mathbf{par}_{\boldsymbol{i}}$, then the matrix is $\boldsymbol{\Delta}_{\boldsymbol{x_k,x_k}}$. Otherwise, the matrix is the identity matrix. In this way this Kronecker product distributes the elements of $\boldsymbol{Q}_{\boldsymbol{X_i}|\mathbf{par}_{\boldsymbol{i}}}$ to the relevant entries of the joint intensity matrix.

We can now define the joint intensity matrix. Let $\tilde{\boldsymbol{Q}}_{\boldsymbol{X_i}}$ be $\sum_{\mathbf{par}_{\boldsymbol{i}}} \tilde{\boldsymbol{Q}}_{\boldsymbol{X_i}|\mathbf{par}_{\boldsymbol{i}}}$. Then, $\boldsymbol{Q} = \sum_i \tilde{\boldsymbol{Q}}_{\boldsymbol{X_i}}$. Figure 15 gives an example for the CTBN of Figure 13. Figure 16 gives another example. While the Kronecker product in general does not handle the diagonal elements, the expansion works for the intensity matrix in this case, since only one of the matrices in each product is non-diagonal.

$$\tilde{\boldsymbol{Q}}_{\boldsymbol{X_1}} = \sum_{x_2} \boldsymbol{Q}_{\boldsymbol{X_1}|\boldsymbol{x_2}} \otimes \boldsymbol{\Delta}_{\boldsymbol{x_2},\boldsymbol{x_2}} \otimes \boldsymbol{I}$$
$$\tilde{\boldsymbol{Q}}_{\boldsymbol{X_2}} = \sum_{x_1} \boldsymbol{\Delta}_{\boldsymbol{x_1},\boldsymbol{x_1}} \otimes \boldsymbol{Q}_{\boldsymbol{X_2}|\boldsymbol{x_1}} \otimes \boldsymbol{I}$$
$$\tilde{\boldsymbol{Q}}_{\boldsymbol{X_3}} = \sum_{x_2} \boldsymbol{I} \otimes \boldsymbol{\Delta}_{\boldsymbol{x_2},\boldsymbol{x_2}} \otimes \boldsymbol{Q}_{\boldsymbol{X_3}|\boldsymbol{x_2}}$$

$$\boldsymbol{Q} = \tilde{\boldsymbol{Q}}_{\boldsymbol{X_1}} + \tilde{\boldsymbol{Q}}_{\boldsymbol{X_2}} + \tilde{\boldsymbol{Q}}_{\boldsymbol{X_3}}$$

Figure 15: Sum of Kronecker encoding of the rate matrix Q of the CTBN in Figure 13.



$$\tilde{\boldsymbol{Q}}_{\boldsymbol{W}} = \boldsymbol{Q}_{\boldsymbol{W}} \otimes \boldsymbol{I} \otimes \boldsymbol{I} \otimes \boldsymbol{I}$$
$$\tilde{\boldsymbol{Q}}_{\boldsymbol{X}} = \sum_{z} \boldsymbol{I} \otimes \boldsymbol{Q}_{\boldsymbol{X}|\boldsymbol{z}} \otimes \boldsymbol{I} \otimes \boldsymbol{\Delta}_{\boldsymbol{z},\boldsymbol{z}}$$
$$\tilde{\boldsymbol{Q}}_{\boldsymbol{Y}} = \sum_{w} \boldsymbol{\Delta}_{\boldsymbol{w},\boldsymbol{w}} \otimes \boldsymbol{I} \otimes \boldsymbol{Q}_{\boldsymbol{Y}|\boldsymbol{w}} \otimes \boldsymbol{I}$$
$$\tilde{\boldsymbol{Q}}_{\boldsymbol{Z}} = \sum_{x,y} \boldsymbol{I} \otimes \boldsymbol{\Delta}_{\boldsymbol{x},\boldsymbol{x}} \otimes \boldsymbol{\Delta}_{\boldsymbol{y},\boldsymbol{y}} \otimes \boldsymbol{Q}_{\boldsymbol{Z}|\boldsymbol{x},\boldsymbol{y}}$$

$$\boldsymbol{Q} = \tilde{\boldsymbol{Q}}_{\boldsymbol{W}} + \tilde{\boldsymbol{Q}}_{\boldsymbol{X}} + \tilde{\boldsymbol{Q}}_{\boldsymbol{Y}} + \tilde{\boldsymbol{Q}}_{\boldsymbol{Z}}$$

Figure 16: Sum of Kronecker product encoding of a CTBN with more than one parent per node.

5.1.3 Connection to Decision Diagrams

The above decomposition of a CTBN into a sum of Kronecker products helps clarify the connection to the edge-valued decision diagrams of the previous section. A CTBN is a particularly structured version of the disjunctive EV*MDD encoding of Section 4.5.1 paired with the identity encoding of Section 4.5.2. In particular, a CTBN describes a CTMP which can also be described by a sum of EV*MDDs in fully-identity-reduced form. The two descriptions have the same order space complexity. The decision-diagram encoding has one EV*MDD for each variable and each joint value for its parents.

Figure 17 shows the disjunction of EV*MDDs for the CTBN in Figure 13. Each EV*MDD only encodes the non-identity matrices in the Kronecker product expression; the identity matrices are implied by the fully-identity-reduced form. As another example, for the CTBN in Figure 16, we could construct 9 EV*MDDs: 1 for W, 2 for each of X and Y, and 4 for Z.

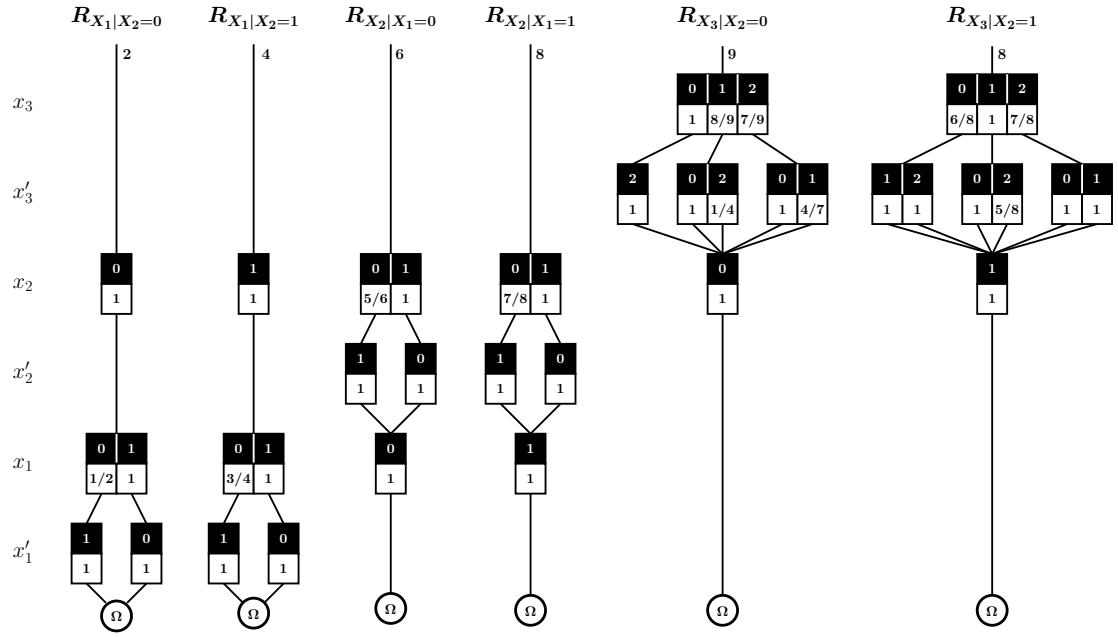


Figure 17: A set of identity-reduced EV*MDDs whose sum is the same as the CTBN of Figure 13.

Note that a disjunction of EV*MDDs can compactly encode structure *within* a variable's local rate matrix, while a CTBN cannot. In this way, they represent a generalization that can exploit context-sensitive independence.

Whether merging EV*MDDs for a given variable or merging EV*MDDs for multiple variables will result in a reduction or increase in the representation size is largely an empirical question. However, we would generally expect an increase in size because only one transition is allowed at a time, so the paths through the levels must remember whether any previous variable has changed and, if so, which one (for example, as in Figure 11).

5.2 Sampling

Sampling from a CTBN can be done by straight-forward application of the sampling method described in Section 2.1. We need not construct the full intensity matrix. Instead, for any joint assignment $\boldsymbol{x}$, we can find the diagonal element by summing the diagonals of the relevant conditional intensity matrices. This gives us the rate of the exponential to sample for the time to the next variable change. We can read the intensities for each variable's potential transitions from the relevant row of its conditional intensity matrix and we select a variable and the new state for that variable in proportion to the intensity. This process takes $O(L)$ time for each transition (where L is the number of variables).

We can do better by exploiting the racing and memoryless properties of exponential distributions (discussed in Section 2.1). To select which variable transitions next, we will race exponential distributions for each variable with rates of the corresponding diagonal

function SampleCTBN(CTBN, initial distribution π_0, end time T)
  Let $\mathcal{T}r \leftarrow$ empty trajectory
  Let $(x_1, x_2, \ldots, x_L) \leftarrow$ joint sample from π_0 ▷ As per algorithm for π_0's representation
  for i from 1 to L **do**
    Add $(X_i = x_i \ @ \ 0)$ to $\mathcal{T}r$
  Let $t \leftarrow 0$
  Let $E \leftarrow$ an empty event priority queue of time-variable pairs.
  repeat
    for all variables X_i that do not have an event in E **do**
      Sample Δt from an exponential with rate $q_{x_i|\mathbf{par_i}}$
      Add $\langle t + \Delta t, X_i \rangle$ to E
    Let $\langle t, X_i \rangle \leftarrow$ the earliest event in E ▷ Update t and get new variable to change
    if $t < T$ **then**
      Sample x_i' from a multinomial proportional to $q_{x_i,x_i'|\mathbf{par_i}}$
      Let $x_i \leftarrow x_i'$ ▷ Update local copy of variable assignments
      Add $(X_i = x_i \ @ \ t)$ to $\mathcal{T}r$
      Remove X_i and all children of X_i from E ▷ Their times must be resampled
  until $t \geq T$

Figure 18: Algorithm to sample from CTBN

elements of their conditional intensity matrices. We then note that if a variable is not chosen, we can treat its transition time as two separate random draws: a draw stating that its transition time is after the chosen time and a draw stating when after the chosen transition time it will next transition (because of the memoryless property of the exponential distribution). That means that if the chosen transition did not affect the rate for the variable in question, we do not need to resample its transition time. By using a priority queue for transitions times (not durations), we can reduce the running time per transition to $O(D \ln L)$ where D is the maximal out-degree of the graph. This method is made explicit in Figure 18.

5.3 Inference

Inference in a CTBN is the process of calculating an expected value of the full trajectory, given some partial trajectory. The most basic case is to infer the conditional probability of a single variable at a single time point (the expectation of an indicator function) given a partial trajectory. There are many ways in which a trajectory may be partial. The most obvious for a variable-based model like a CTBN is to have variables only observed at particular times and intervals. Each variable can have its own observation times and intervals. Thus, for each variable, we assume we have evidence like that of Section 2.5.1: There are time points at which the variable has known values and there are time intervals during which the variable has known values (which might include observations of transitions).

Unfortunately, even if there is no evidence, this problem is NP-hard. In particular, deciding whether the marginal probability of a single value of a single variable at a single

time point is greater than any positive threshold is NP-hard. This has been generally accepted, although never formally demonstrated. We provide the proof in the Appendix.

Thus, all known algorithms for CTBN inference have exponential (in the number of variables) running time. The simplest method is to treat the CTBN as a general CTMP with a single intensity matrix $\boldsymbol{Q}$. We can apply the forward and backward passes of Section 2.5. While the intensity matrix can be stored in compact form, the resulting vectors require space for each instantiation to every variable in the CTBN (exponential space). We need only keep values for states consistent with the evidence. Thus, if at all times only a few variables are unobserved, the inference is tractable. But, if there are periods during which many variables are unobserved, we require approximate inference methods (overviewed in Section 5.5).

Other than calculating the probability of a variable at a time, the other common case of inference is to calculate the expected sufficient statistics. As shown in Section 5.4.1, this means calculating $\bar{N}[x_i, x_i'|\mathbf{par_i}]$ and $\bar{T}[x_i|\mathbf{par_i}]$ for all values of i, x_i, x_i', and $\mathbf{par_i}$. The former is the expected number of times variable X_i transitioned from x_i to x_i' while its parents were in state $\mathbf{par_i}$, and the latter is the expected amount of time variable X_i was in state x_i while its parents were in state $\mathbf{par_i}$.

The proof for marginal calculation can easily be adapted to show that deciding whether these quantities are non-zero is also NP-hard. Therefore, the only known method is to again treat the system as a general CTMP with a single large Q matrix. We can then apply Equation 65 and Equation 64 to find the expected number of transitions and expected amount of time for any joint assignments. If we let $J(x_i, \mathbf{par_i})$ be the set of joint assignments to all variables that are consistent with $X_i = x_i$ and $\mathrm{Par}_i = \mathbf{par_i}$, we can find the expected sufficient statistics for the CTBN as

$$\bar{T}[x_i|\mathbf{par_i}] = \sum_{\boldsymbol{x} \in J(x_i, \mathbf{par_i})} \bar{T}[\boldsymbol{x}] \tag{80}$$

$$\bar{N}[x_i, x_i'|\mathbf{par_i}] = \sum_{\boldsymbol{x} \in J(x_i, \mathbf{par_i})} \sum_{\boldsymbol{x'} \in J(x_i', \mathbf{par_i})} \bar{N}[\boldsymbol{x}, \boldsymbol{x'}] \tag{81}$$

5.4 Parameter and Graph Estimation

The initial distribution of a CTBN can be estimated separately using any standard method for estimation of a Bayesian network (or whatever other compact representation is desired). This requires only data about the value of the trajectory's value (or trajectories' values) at time 0.

We will concentrate on estimation of the rate parameters and dynamics graph structure ($\mathcal{G}$). This exposition will assume there is a single trajectory, $\mathcal{T}r$. However, multiple trajectories can be used by summing their sufficient statistics.

5.4.1 Parameter Estimation

The set of CTBNs with a fixed graph structure is just a subset of the exponential family of CTMPs in which most parameters are fixed to 0 and many of the remaining ones are tied to each other (share the same value). Thus, the log-likelihood of Equation 30 applies here

too, but where the sufficient statistics for tied parameters are summed:

$$\ln p_{\text{CTBN}}(\mathcal{T}\!r) = \ln P_0(\mathcal{T}\!r(0)) + \sum_{i,\mathbf{par_i},x_i} \left(-T[x_i|\mathbf{par_i}] q_{x_i|\mathbf{par_i}} + \sum_{x_i' \neq x_i} N[x_i, x_i'|\mathbf{par_i}] \ln q_{x_i,x_i'|\mathbf{par_i}} \right) \tag{82}$$

$$= \ln P_0(\mathcal{T}\!r(0)) + \sum_{i,\mathbf{par_i},x_i,x_i' \neq x_i} \left(-T[x_i|\mathbf{par_i}] q_{x_i,x_i'|\mathbf{par_i}} + N[x_i, x_i'|\mathbf{par_i}] \ln q_{x_i,x_i'|\mathbf{par_i}} \right) \tag{83}$$

$$= \ln P_0(\mathcal{T}\!r(0)) + \sum_i l_{X_i}(\mathcal{T}\!r) \tag{84}$$

where i ranges over variables, $\mathbf{par_i}$ ranges over joint assignments to the parents of i, and x_i and x_i' range over differing assignments to X_i. $T[x_i|\mathbf{par_i}]$ denotes the amount of time $X_i = x_i$ while $\text{Par}_i = \mathbf{par_i}$. Similarly, $N[x_i, x_i'|\mathbf{par_i}]$ denotes the number of transitions of X_i from x_i to x_i' while $\text{Par}_i = \mathbf{par_i}$. These new sufficient statistics are sums of the sufficient statistics of the flat CTMP, summing over all assignments to all CTBN variables in which X_i and Par_i remain the same (see Equations 80 and 81). Given a complete trajectory, we can construct them directly without employing such (exponentially large) sums. The last line above is by definition of l_{X_i}, the "local log-likelihood" of variable X_i. Note that this is a function only of the trajectories of X_i and its parents (not of all of $\mathcal{T}\!r$).

Maximizing Equation 83 is a straight-forward extension of maximizing Equation 30:

$$\hat{q}_{x_i,x_i'|\mathbf{par_i}} = N[x_i, x_i'|\mathbf{par_i}]/T[x_i|\mathbf{par_i}] \ . \tag{85}$$

We can produce Bayesian posterior distributions over the parameters if we take independent conjugate prior distributions over each $q_{x_i,x_i'|\mathbf{par_i}}$ parameter (Nodelman et al., 2003). Just as for a flat CTMP, our conjugate prior is a gamma distribution with hyper-parameters $\alpha_{x_i,x_i'|\mathbf{par_i}}$ and $\tau_{x_i,x_i'|\mathbf{par_i}}$ for parameter $q_{x_i,x_i'|\mathbf{par_i}}$. The resulting posterior is also a gamma distribution with corresponding hyper-parameters $\alpha_{x_i,x_i'|\mathbf{par_i}} + N[x_i, x_i'|\mathbf{par_i}]$ and $\tau_{x_i,x_i'|\mathbf{par_i}} + T[x_i|\mathbf{par_i}]$. Thus the MAP parameter estimates are

$$\hat{q}_{x_i,x_i'|\mathbf{par_i}} = \frac{N[x_i, x_i'|\mathbf{par_i}] + \alpha_{x_i,x_i'|\mathbf{par_i}}}{T[x_i|\mathbf{par_i}] + \tau_{x_i,x_i'|\mathbf{par_i}}} \ . \tag{86}$$

5.4.2 Structure Estimation

Estimating the CTBN structure could be accomplished by statistical tests of the independence of the processes. Yet, we are unaware of any methods that use this or of suitable independence tests.

Instead, CTBN structures have been estimated by graph scoring functions. If the score function decomposes as the likelihood does (Equation 83) into a sum of terms, one per variable, in which the selection of a variable's parents only affects the term for the same variable, the search for the maximal scoring graph is very simple. Each variable's parents can be chosen independently by maximizing the corresponding term in the sum. While there are an exponentially large (in the total number of variables) number of parent sets

to consider for each variable, if we limit the cardinality of parent sets to no more than D, then each variable's parents can be chosen by exhaustive search and the total running time is $O(L2^D)$, which is linear in the number of variables, L.

This is in contrast to Bayesian networks where a similar strategy does not lead to an efficient algorithm (unless a variable ordering is known *a priori*). Learning CTBNs structure is efficient because there are no restrictions on the graph: A CTBN's graph may be cyclic. A similar situation arises with dynamic Bayesian networks (DBNs). If we only allow inter-time-slice edges (those from the "previous" time point to the "current" time point), the graph structure may be searched efficiently, like in CTBNs. However, if we allow intra-time-slice edges (those within the "current" time point) in a DBN, we must enforce acyclicity constraints and the search is no longer efficient.

The Bayesian information criterion (Lam & Bacchus, 1994) can be made into a score:

$$score_{BIC}(\mathcal{G} : \mathcal{T}r) = \left(\sum_i l_{X_i}(\mathcal{T}r)\right) - \frac{\ln|\mathcal{T}r|}{2} Dim(\mathcal{G}) \tag{87}$$

$$= \sum_i \left(l_{X_i}(\mathcal{T}r) - \frac{\ln|\mathcal{T}r|}{2}\left|\boldsymbol{Q}_{\boldsymbol{X_i}|\mathbf{par_i}}\right|\right) \tag{88}$$

where $Dim(\mathcal{G})$ is the number of independent parameters in the network defined by the graph $\mathcal{G}$ and $\left|\boldsymbol{Q}_{\boldsymbol{X_i}|\mathbf{par_i}}\right|$ is the number of independent parameters in the conditional intensity matrices associated with X_i. This second term is equal to $n_i(n_i-1)$ (because the diagonal elements are not independent) times the number of parent instantiations. The data size, $|\mathcal{T}r|$, is the number of transitions in the trajectory $\mathcal{T}r$ (or in the total data set if it consists of multiple trajectories). This score is consistent (Nodelman et al., 2003) because the term $l_{X_i}(\mathcal{T}r)$ grows linearly with the amount of data and represents the likelihood and the second term grows logarithmically with the amount of data and penalizes excess parameters.

A Bayesian score can also be constructed by placing a prior on graphs (as well as parameters) and finding the maximum of $\ln P(\mathcal{G} \mid \mathcal{T}r) = \ln p(\mathcal{T}r \mid \mathcal{G}) + \ln P(\mathcal{G}) - \ln p(\mathcal{T}r)$. The last term isn't affected by the choice of $\mathcal{G}$, so we drop it. We assume structure modularity: $\ln P(\mathcal{G}) = \sum_i \ln P(\mathrm{Par}_i)$. The remaining data term, $\ln P(\mathcal{T}r \mid \mathcal{G})$, is the (logarithm of the) integral of the likelihood multiplied by the prior, over all possible parameter values. Using the independent gamma priors from above, this decomposes into a separate term for each variable (dropping the $\ln P_0(\mathcal{T}r(0))$ term which does not affect the choice of $\mathcal{G}$):

$$\ln\, p(\mathcal{T}r \mid \mathcal{G}) = \sum_{i,\mathbf{par_i},x_i \neq x_i'} \ln \int_0^\infty \frac{(\tau_{x_i,x_i'|\mathbf{par_i}})^{\alpha_{x_i,x_i'|\mathbf{par_i}}+1}}{\Gamma(\alpha_{x_i,x_i'|\mathbf{par_i}}+1)} \exp\Big[-(T[x_i|\mathbf{par_i}] + \tau_{x_i,x_i'|\mathbf{par_i}}) q_{x_i,x_i'|\mathbf{par_i}}$$

$$+ (N[x_i,x_i'|\mathbf{par_i}] + \alpha_{x_i,x_i'|\mathbf{par_i}}) \ln q_{x_i,x_i'|\mathbf{par_i}}\Big]\, dq_{x_i,x_i'|\mathbf{par_i}} \tag{89}$$

$$= \sum_i \sum_{\mathbf{par_i},x_i \neq x_i'} \ln\left(\frac{(\tau_{x_i,x_i'|\mathbf{par_i}})^{\alpha_{x_i,x_i'|\mathbf{par_i}}+1}}{\Gamma(\alpha_{x_i,x_i'|\mathbf{par_i}}+1)} \frac{\Gamma(N[x_i,x_i'|\mathbf{par_i}] + \alpha_{x_i,x_i'|\mathbf{par_i}} + 1)}{(T[x_i|\mathbf{par_i}] + \tau_{x_i,x_i'|\mathbf{par_i}})^{N[x_i,x_i'|\mathbf{par_i}]+\alpha_{x_i,x_i'|\mathbf{par_i}}+1}}\right) \tag{90}$$

$$= \sum_i lscore_B(\mathbf{par_i} : \mathcal{T}r) \tag{91}$$

where the last line is by the definition of $lscore_B$. This derivation is almost the same as the one given in Nodelman et al. (2003). The difference is that our prior consists of a gamma distribution for each independent variable whereas their prior consists of a gamma distribution for each diagonal rate parameter and a Dirichlet prior over the ratios $q_{x_i,x_i'|\mathbf{par}_i}/q_{x_i|\mathbf{par}_i}$. The two are equivalent, but parameterized differently.

The Bayesian score is therefore

$$score_B(\mathcal{G} : \mathcal{T}r) = \sum_i lscore_B(\mathbf{par}_i : \mathcal{T}r) + \ln P(\mathrm{Par}_i) \ . \tag{92}$$

It converges to the BIC score in the limit of infinite data (Nodelman et al., 2003) and is therefore also consistent.

5.4.3 Incomplete Data

For the case where the trajectory $\mathcal{T}r$ is incomplete, we are back in the same situation as in Section 2.6. As the likelihood takes the same form in a CTBN as in a general CTMP, the solutions for maximizing the likelihood of this incomplete trajectory have the same form. Namely, if we compute the expected sufficient statistics (using inference), we can apply gradient ascent or expectation-maximization to find the maximum likelihood parameters. The gradient is

$$\frac{\partial p(\mathcal{T}r)}{\partial q_{x_i,x_i'|\mathbf{par}_i}} = p(\mathcal{T}r)\left(\frac{\bar{N}[x_i, x_i'|\mathbf{par}_i]}{q_{x_i,x_i'|\mathbf{par}_i}} - \bar{T}[x_i|\mathbf{par}_i]\right) \tag{93}$$

and the expectation-maximization update equation is the same as Equation 85 except the sufficient statistics are replaced by their expected values, given the partially observed trajectory and the current model.

For graph estimation, we can apply structural expectation-maximization (Friedman, 1997) (SEM) to CTBNs (Nodelman, Shelton, & Koller, 2005). While SEM for Bayesian networks can be a little complex due to the structure search step, for CTBNs, it is simpler as the structure search step need not enforce acyclicity constraints and therefore can be carried out more simply (see above). The tricky point (which also holds for standard Bayesian networks) is that the graph search scoring function must be calculated using expected sufficient statistics and therefore, given a current model, our inference algorithm must produce expected sufficient statistics not only for the current model's parent sets, but also for any other parent sets to be considered by the structure search. If using exact inference (by flattening the CTBN to a general CTMP), these are available. However, approximate methods (below) differ in how simple it is to extract such expected sufficient statistics. Once this inference is performed, a joint optimization of parameters and structure is performed, the new model is used to find new expected sufficient statistics, and the process repeats.

5.5 Approximate Inference

As mentioned in Section 5.3, exact inference is intractable when there are many concurrently missing variables. Therefore, many approximate inference methods have been developed. We briefly cover them in this section, but would refer to the full papers for more complete descriptions.

5.5.1 Sampling-Based Inference

Sampling is an obvious method for producing approximate inference. It has a number of advantages. First, it produces a set of full trajectories from which any inference question can be answered. Second, most sampling methods converge to the correct value if allowed to run long enough. Third, sampling methods are usually easily parallelized, lending themselves to multiple processors and multiple cores.

Hobolth and Stone (2009) have a description of a number of such methods for the unstructured case and for full evidence at the beginning and the end of the trajectory, but no evidence in between. Here we discuss work on CTBNs and for more general evidence patterns.

Fan and Shelton (2008) and Fan, Xu, and Shelton (2010) developed an importance sampler and a particle filter and smoother. Forward sampling (like in Figure 18) can be turned into an importance sampler by taking the observed data as given and sampling for the missing portions, marching time along. The weight of the sample is the probability of having sampled the observed data (which was not sampled) given the trajectory up to those data. The problem arises when a variable goes from being not observed to being observed. In this case, the sampling must agree with "up coming" observation evidence. Adding a transition exactly when the evidence starts is not correct (as there is almost surely not an event at that time). These samplers handle it with some forward look ahead to sample the necessary transition in advance, with suitable importance weight corrections. This is then extended to a particle filter and smoother which are resampled based on the number of transitions, rather than the absolute time. This method was extended to more general temporal models by Pfeffer (2009).

El-Hay, Friedman, and Kupferman (2008) developed a Gibbs sampler for CTBN models. They start with a simply developed trajectory that agrees with all of the evidence. Then, the algorithm removes a single variable's full trajectory and resamples it (keeping any time periods during which the value is known). Conditioned on the full trajectories of a variable's Markov blanket (the union of the variable's children, parents, and children's parents), the trajectory for that variable is independent of all other variables, so the sampler needs to only consider the variable's Markov blanket. The posterior distributions over the times of transitions are no longer exponential distributions. Their forms are complex and thus the Gibbs sampler must sample by performing binary search.

Fan and Shelton (2009) combined the ideas from the Gibbs sampler and their earlier work on importance sampling to produce a Metropolis-Hastings sampler. The importance sampling method is used instead of Gibbs sampling and the importance weight is used to find the acceptance probability. While faster to generate samples, the samples take longer to converge. The balance of the trade-off depends on the typicality of the evidence and the inference query.

Rao and Teh (2011, 2013) used uniformization to develop an auxiliary Gibbs sampler which is faster than the previous Gibbs sampler. The auxiliary variables are the times of the self-transitions from an uniformization sampler (Section 2.3). Thus to resample a variable, the algorithm samples auxiliary times, given the old trajectory (which can be done quickly). It then throws away all of the transitions, but keeps the full set of times (old times and the new times). Then, using a forward-backward two-pass algorithm, state

transitions are sampled from the uniformized discrete-time process (conditioned on the evidence). Finally, the self transitions are discarded. Rao and Teh (2012) extended this to a time-varying uniformization rate to speed up convergence, but only explicitly in the case of an unstructured process.

5.5.2 Non-Sampling Methods

A number of other non-sampling methods have also been proposed. Their advantages include determinism (often helpful when used inside of EM to keep the estimates consistent), and fewer parameters that need to be set well (number of samples, length of burn-in, and others).

Cohn, El-Hay, Kupferman, and Friedman (2009) and Cohn, El-Hay, Friedman, and Kupferman (2010) derived a mean-field approximation. The approximate distribution is an independent *time-inhomogeneous* Markov process for each variable. That is, the variables are independent (in the approximation), but the intensities depend on time. The natural parameterization also differs slightly. Instead of transition rates, transition densities are used, but the idea is the same. The resulting algorithm changes one variable's distribution at a time, again depending only on the Markov blanket. The update involves solving a system of differential equations (to get the time-varying parameters of the inhomogeneous Markov process). These are solved by adaptive integration which means that less computation is required during intervals of less rapid change. The result is that the time-varying parameters are represented by a series of time-value points (those produced during adaptive integration) and linear interpolation between these points.

Nodelman, Koller, and Shelton (2005) derived an expectation-propagation method. The propagation uses piece-wise constant time-homogeneous Markov processes, where each piece corresponds to a period of constant evidence. These piece-wise constant approximations are propagated instead of the true marginals (as such marginals would be intractably large). Saria, Nodelman, and Koller (2007) extended this method to subdivide the pieces of the approximations further and adaptively. El-Hay, Cohn, Friedman, and Kupferman (2010) produced a belief propagation algorithm in the same spirit as the mean-field approximation above, employing a free energy functional for CTMPs. Instead of propagating piece-wise time-homogeneous Markov processes, they propagate a single time-inhomogeneous Markov process and then use the same adaptive integration representation as in mean-field. The result is more adaptive and mathematically cleaner.

Finally, for filtering (but not general inference), Celikkaya, Shelton, and Lam (2011) developed a factored version of a uniformized Taylor expansion to approximate the matrix exponential calculations. The result is something similar to that of Boyen and Koller (1998) for dynamic Bayesian networks, but also involving a truncation of an infinite sum and a mixture of propagation distributions. This method is the only current non-sampling method with accuracy bounds, although they are very loose.

5.6 Extensions

As shown above, a CTBN can be converted into a sum of decision diagrams. In that way decision diagrams (and other similarly convertible models) can be viewed as extensions of

CTBNs. Many of the non-Markovian processes of Section 1.1 could, if restricted in the right way, also be so viewed. However, there are a few more direct extensions of CTBNs.

First, El-Hay et al. (2010) introduced continuous-time Markov networks (CTMNs). They are *undirected* graphical models of Markov processes in the same way that CTBNs are *directed* graphical models. They model the subclass of reversible processes, ones for which detailed balance holds: There exists a distribution π (the stationary distribution of the process) such that $\pi(\boldsymbol{x})q_{\boldsymbol{x},\boldsymbol{x'}} = \pi(\boldsymbol{x'})q_{\boldsymbol{x'},\boldsymbol{x}}$ for all pairs of states $\boldsymbol{x}$ and $\boldsymbol{x'}$. A CTMN can be converted to a CTBN by replacing each undirected edge with a pair of directed edges. Their parameterization directly reveals the stationary distribution of the process as a Markov network.

Second, Portinale and Codetta-Raiteri (2009) and Codetta-Raiteri and Portinale (2010) showed an extension of CTBNs to allow for simultaneous transition of multiple variables. It is based on Petri nets and encodes "cascades" of transitions that all happen simultaneously.

Finally, Weiss, Natarajan, and Page (2012) presented a method for constructing the local rate matrices for each variable not as a matrix, but as the multiplication of regression trees. This is akin to exploiting context-specific independence (Shimony, 1991) in a standard Bayesian network by use of trees (Boutilier, Friedman, Goldszmidt, & Koller, 1996). This multiplication of trees is not the same as our reduction of a CTBN to a sum of EV*MDDs (see Figure 17). In particular, their trees do not require the tests be made in a particular variable order, they use trees instead of DAGS, and they multiply the trees together (instead of adding them). Weiss et al. (2012) also give a boosting-style algorithm for learning this parameterization. No similar method is known for learning a sum of EV*MDDs.

6. Applications and Current Directions

To provide some context for the theory and algorithms above, we describe how these methods have been used in applications. We then discuss what we believe to be the most promising and pressing research directions.

6.1 Decision-Diagram-Based Models

Structured CTMPs arise in many applications areas, from performance and reliability evaluation of computer systems to the investigation of biological systems. As the underlying CTMPs describing the dynamics being analyzed are usually very large, most software tools used for such studies rely on compact symbolic techniques to encode them.

In particular, PRISM (Kwiatkowska et al., 2011) uses a hybrid form of MTBDDs, Möbius (Deavours et al., 2002) uses Matrix Diagrams (a data structure almost equivalent to EV*MDDs), and SMART (Ciardo et al., 2006) uses EV*MDDs to encode the transition rate matrix of a CTMP. These tools can compute both stationary and transient exact numerical solutions of compactly encoded CTMPs. (Indeed, they can compute much more complex *stochastic temporal logic properties* such as those that can be expressed in CSL (Baier, Haverkort, Hermanns, & Katoen, 2000), but these, too, ultimately require a sequence of stationary or transient numerical solutions.) While such exact solutions place very large computational demands due to the exponential explosion of the state space, the situation is often somewhat mitigated by the fact that, in most applications targeted by these tools, the actual state space is a small subset of the full cross-product of the state variable values.

As an example application, we briefly summarize a study done using PRISM for the analysis of a complex biological pathway called FGF (Fibroblast Growth Factor) (Heath, Kwiatkowska, Norman, Parker, & Tymchyshyn, 2006). The state of the system consists of the number of proteins (e.g., A, B) or protein complexes (e.g., $A{:}B$) present at the current time. The events in the system consist of various reactions such as *complexation* (e.g., $A + B \to A : B$) and *decomplexation* (its reverse, $A{:}B \to A + B$), as well as *degradation* (e.g., $A \to$). Finally, each event has an occurrence rate, which can of course depend on the number of proteins currently present for the types involved in the particular reaction. The actual model for the FGF pathway, even after substantial simplifications to focus only on key and well-known aspects in real cells, contains 87 different proteins or protein complexes (each of them corresponding to a local state variable) and 50 different reactions (if we count complexation and decomplexation separately). We stress that each reaction concerns just a few proteins or compounds; thus its decision diagram representation in isolation is quite compact.

Even the smallest meaningful model where there is only zero or one protein or compound of each type would have a potential state space of size 2^{87}. However, as almost always the case in this type of models, only a tiny fraction of these states is reachable, thus the model used in the work of Heath et al. (2006) has merely 801,616 states and 560,000 state-to-state transitions. The study focused on several key questions such as what fraction of time particular proteins are bound, or the probability that a particular degradation has occurred within a given time bound, all quantities obtainable through numerical stationary or transient analysis of the underlying CTMP. Notwithstanding the relatively small size of the state space (which could likely be scaled by a factor of 1000, to around 10^8 states, given a modern workstation with sufficient memory) the results and predictions obtained from this model using PRISM were shown to agree with biological data, demonstrating the viability of these technique to perform "in silico genetics" as a much less costly alternative to the "in vitro" experiments traditionally performed in biology.

6.2 Continuous-Time Bayesian networks

CTBNs have been employed on a number of real-world datasets and problems including life event history data (Nodelman et al., 2003), user activity modeling (Nodelman & Horvitz, 2003), computer system failure modeling (Herbrich, Graepel, & Murphy, 2007), mobile robotics (Ng, Pfeffer, & Dearden, 2005), network intrusion detection (Xu & Shelton, 2008, 2010), phylogentic trees (Cohn et al., 2009), social networks (Fan & Shelton, 2009), cardiovascular health model (Weiss et al., 2012), and heart failure (Gatti, Luciani, & Stella, 2012). Many of these also innovated in extending the CTBN framework. For instance, Ng et al. (2005) allowed for continuous-state variables whose dynamics are dictated by differential equations. The form of evidence is more limited, but a particle filter is developed for this situation. Cohn et al. (2009) applied the CTBN model to a "time-tree" to allow branching (as first done in Felsenstein, 1981 for general CTMPs). Weiss et al. (2012) added context-specific independence.

To give an idea of the application of CTBNs, we briefly review the intrusion detection work of Xu and Shelton (2008, 2010). In this work, the goal was to build a model of normal

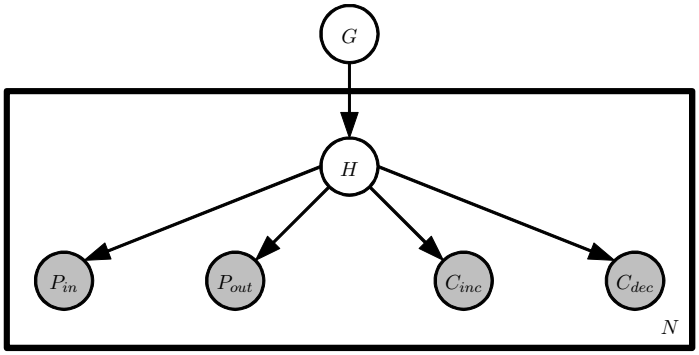


Figure 19: CTBN model for network traffic (Xu & Shelton, 2010). N is the number of destination ports.

computer system events, specific to a particular machine. This model could then be used to detect abnormal events or time windows as a method of intrusion detection.

Two models were built: one modeling network traffic in and out of the machine, and one modeling system calls from processes. The network model was as in Figure 19. There is one global hidden variable G with four states. The traffic is divided into different destination ports (for instance, 80 for HTTP traffic and 995 for POP traffic). The most frequent eight ports are separated out and the traffic from the remaining destinations are grouped together. Each of these $N = 9$ groups has its own model (the plate in Figure 19). This submodel has one hidden variable H and four completely observed binary variables, P_{in}, P_{out}, C_{inc}, and C_{dec}, representing packets sent and received and connections started and stopped respectively. These observed variables toggle state to represent an event of the relevant type, but their state has no intrinsic meaning. Therefore, their matrices have only a single independent parameter: the rate of transition from either state to the other. In this way, these observed variables are really conditional Poisson processes.

The hidden variable H is structured to exploit domain knowledge. It has 8 states which are grouped into pairs, one pair for each of its children. For each pair only one of its children has a non-zero rate. Thus, H encodes what type of event will happen and some substate of this meta-state.

The entire model has 4×8^9 or approximately 500 million hidden states (as the observed variables are observed at all times, only the distribution over the G and the Hs need be tracked). Yet, each submodel has only 8 hidden states, so exact inference over a submodel is very reasonable. Thus, they adapted the particle filter and smoother of Fan and Shelton (2008) to distributional particles, producing a Rao-Blackwellized particle filter in which G is sampled and each of the models (which are independent, given a full trajectory of G) are reasoned about exactly.

This inference method allows learning of specific models to each host using EM. These models were then run on data in which computer virus or worm traffic had been injected (very slowly to make it blend in with background traffic). The model was asked the likelihood of 50-second window of traffic (given the previous traffic). This likelihood was thresh-

olded to produce an alarm (if the likelihood dropped too low). The results out-performed SVM-spectrum kernels, nearest-neighbor (using features from the computer network literature for this task), and other methods on this task and other similar tasks.

For process system calls, the model was similar, with a single hidden variable coordinating the behavior of a set of observed system-call variables. The dataset on which this model was trained had time stamps for each system call. However, due to clock resolution, many time stamps were the same. Yet, the temporal order was preserved (although the exact durations between events was not). The paper demonstrates a method to use such data, without assuming event durations, but employing the ordering. In this case, the results were better than the SVM-spectrum kernel and nearest-neighbor and the same as stide with frequency thresholding (Warrender, Forrest, & Pearlmutter, 1999).

6.3 Relative Comparison

Neither of the above two applications could currently be tackled with the other modeling language. In the biological pathway example of Section 6.1, a transition in the system involved more than one variable (increasing the number of protein complexes, while decreasing the number of individual proteins). A CTBN cannot represent simultaneous transitions of multiple variables. The pathways cannot be reformulated in terms of composite variables to prevent such simultaneous transitions without placing all of the state in a single variable.

As a simpler example, consider a system of three variables, x, y, and z. A single event performs three variable updates at the same time: $\{x' \Leftarrow x+y; y' \Leftarrow y+1; z' \Leftarrow z-1\}$ with rate $r(x,y,z)$. We assume that each variable is a natural number in the range $[0,\ldots,n]$ and that any update that would move a variable outside its range is disabled. Figure 20 demonstrates how an EV*MDD can encode such a transition. Neither a Kronecker encoding nor a CTBN can encode this event without merging variables.

Likewise, the network traffic example from Section 6.2 cannot be handled with current decision-diagram-based models. It depends on hidden unseen variables and estimation of transient solutions conditioned on data. More critically, it relies on estimation of the model parameters from data, which has not been developed for decision-diagram models.

6.4 Current Research Directions

There are a range of open modeling, algorithmic, and theoretic problems. First, questions of steady-state distributions and efficient exact solutions have not been addressed for CTBNs (as they have for EV*MDDs). Similarly, questions of structure and parameter estimation and approximate inference have not been addressed for EV*MDDs (as they have for CTBNs).

Optimal decision making has been formulated as general continuous-time Markov decision processes (Puterman, 1994). Yet, extending the general mathematical framework to structured variable-based models is largely unexplored (Kan & Shelton, 2008).

CTBNs were extended to handle continuous-valued variables and measurements in a limited fashion (Ng et al., 2005), but otherwise this has been unexplored. For many applications this is critical. If the underlying system is discrete and the measurements are continuous, techniques like those in Section 2.5 work. But, systems with continuous state require stochastic differential equations (Øksendal, 2003), at least in some form. The work

x	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
y	0	0	0	1	1	1	2	2	2	0	0	0	1	1	1	2	2	2	0	0	0	1	1	1	2	2	2
z	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2
x'	–	–	–	–	–	–	–	–	–	0	1	2	1	2	–	–	–	–	0	1	2	1	2	–	–	–	–
y'	–	–	–	–	–	–	–	–	–	1	1	1	2	2	–	–	–	–	1	1	1	2	2	–	–	–	–
z'	–	–	–	–	–	–	–	–	–	0	0	0	0	0	–	–	–	–	1	1	1	1	1	–	–	–	–

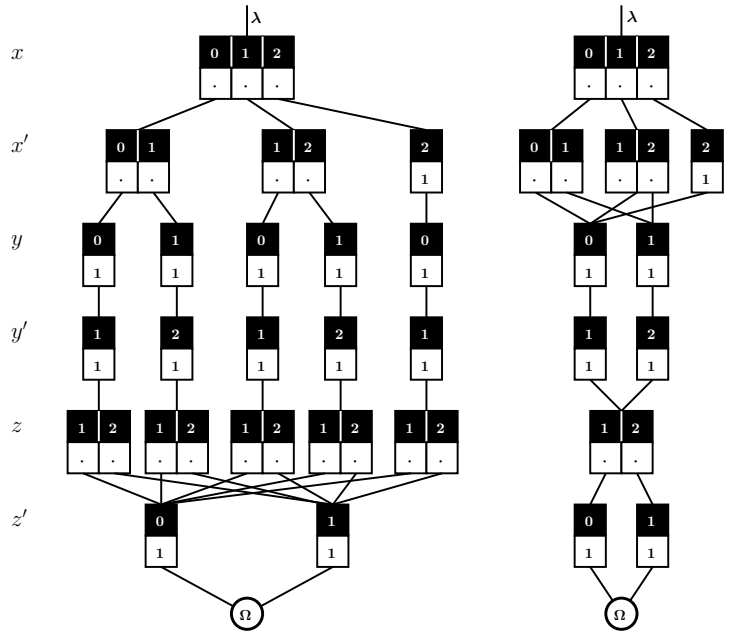


Figure 20: An example of an EV*MDD encoding simultaneous transitions of multiple variables. Top: the 10 possible transitions and resulting states (dash indicates disabled) from state (x, y, z) to state (x', y', z'). Bottom, left: worst case for an arbitrary set of 10 rates for $r(x, y, z)$. Bottom, right: best case when $r(x, y, z) = r_1(x, y)r_2(z)$. A dot indicates a positive value (used to encode the particular rates). In each block of dots, one must be equal to 1 and the others must be ≤ 1.

of Särkkä (2006) describes filtering and smoothing in such models. Yet, parameter estimation is much more difficult and systems with both continuous and discrete state variables have not been systematically addressed.

Finally, new approximate inference methods are always of interest (as with any probabilistic model). Recent methods such as the auxiliary Gibbs sampler (Rao & Teh, 2013) and belief propagation (El-Hay et al., 2010) demonstrate that exploiting the properties of continuous time can lead to great benefits. We hope that further research explores more such methods.

7. Conclusions

Compared with discrete-time models, CTMPs are better suited for domains in which data have real-valued time stamps (the time between events is not regular or well-approximated by a single "clock step rate"). Thus, in selecting a value for the time-slice-width (Δt) for a discrete-time model, either the time width will be large resulting in multiple events per time window (obscuring temporal information), or it will be small resulting in unnecessary computational burdens (propagating across many time windows). Further, the optimal middle ground between too large and too small will differ depending on the data size, the application, and the model component.

We have presented two different CTMP modeling languages. Edge-valued decision diagrams (of Section 4) are more general: They allow multiple variables to change simultaneously. By contrast, CTBNs directly encode independence assumptions (see Section 5). Either an EV*MDD or a CTBN might be more compact for a given situation, although any CTBN can be compactly rerepresented as a *sum* of EV*MDDs (see Section 5.1.3).

The models' forms and histories have given rise to differences in available algorithms, and here the distinctions are greater. The literature on exact solution methods is richer for decision diagram models. Furthermore, this literature is more focused on computing the model's steady state. Approximate methods (especially for transients) and model estimation are notably absent (from an artificial intelligence point-of-view). The literature for CTBNs is more focused on model estimation and approximate inference conditioned on evidence. The CTBN literature has paid no attention to issues of reachability (when much of the joint state space is not reachable) and optimization of exact inference methods.

For processes with a natural synchronization clock (such as modeling daily high and low temperatures), a discrete-time model is the best fit. For processes without such a natural time-slice-width we recommend a continuous-time model. If the questions of interest are about steady states of the system or an exact solution is necessary, an EV*MDD is probably the best choice. If the model must be built from data or approximate inference (especially conditioned on data) is necessary, a CTBN is probably the best choice.

However, we have shown that the two models share much in common. Thus, we hope that the efficient exact algorithms from EV*MDDs can be applied to CTBNs and the approximate inference and model estimation methods from CTBNs can be applied to EV*MDDs. If so, then the choice of model would depend more upon the model properties and not the existing suite of algorithms. In particular, a CTBN makes the assumption that each variable is distinct. In contrast, a disjunctive EV*MDD encoding decomposes the system into local events. The variable-level independencies are more easily read from a CTBN graph, but they disallow simultaneous transition of multiple variables. The application domain should guide whether variable-level explicit independences or simultaneous transitions are more important.

Regardless of the model used, we believe time is a continuous quantity and best modeled as such. While the introduction of the matrix exponential would at first seem to complicate matters (compared with discrete time), we believe it makes the true coupling of variables more obvious and opens up mathematical and algorithmic possibilities for more efficient and precise solutions.

Acknowledgements

Shelton was supported by DARPA award FA8750-14-2-0010 and by The Laura P. and Leland K. Whittier Virtual PICU at Children's Hospital Los Angeles (awards UCR-12101024 and 8220-SGNZZ0777-00). Ciardo was supported by the NSF through grant CCF-1442586.

Appendix A. NP-hardness of CTBN Inference

The theorem and proof of the NP-hardness of CTBN inference are straight-forward extensions of the similar proof for Bayesian networks (Koller & Friedman, 2009). The literature has widely accepted it to be true, but no proof has been formally presented. Thus, while straight-forward, we present it here for completeness.

Definition 1. CTBN-Inf *is the following decision problem. Given a CTBN specified as a directed graph $\mathcal{G}$ over nodes $\{X_1, X_2, \ldots, X_L\}$, a set of conditional intensity matrices $\mathcal{Q} = \{\boldsymbol{Q}_{\boldsymbol{X_i}|\mathbf{par}_i}\}$, and an initial distribution π in which each node is independent with marginals $\{\pi_{X_i}\}$; a variable X_j; a value for X_j, x_j; and a time $t > 0$, decide whether $P_{\mathcal{G},\mathcal{Q},\pi}(X_j(t) = x_j) > 0$.*

Theorem 1. ***CTBN-Inf*** *is NP-hard.*

Proof. The proof is a polynomial time reduction from 3-SAT, following the same lines as the similar proof for Bayesian networks.

Given a 3-SAT problem with variables $z_1, z_2, \ldots, z_m$ and clauses $c_1, c_2, \ldots, c_k$ in which $z_{A(i,j)}$ is the jth variable ($j \in \{1, 2, 3\}$) in clause i ($i \in \{1, 2, \ldots, k\}$) with sign $s_{A(i,j)} \in \{+, -\}$, we construct a CTBN with $m + 2k - 1$ binary variables (taking values F or T): $Y_1, Y_2, \ldots, Y_m$, $C_1, C_2, \ldots, C_k$, $B_1, B_2, \ldots, B_{k-2}$, and S.

Variable Y_i has no parents, a uniform initial distribution π_{Y_i}, and an intensity matrix $\boldsymbol{Q}_{\boldsymbol{Y_i}|\emptyset}$ that is all 0.

Variable C_i has three parents: $Y_{A(i,1)}$, $Y_{A(i,2)}$, and $Y_{A(i,3)}$. If none of the truth value of the parents $(y_{A(i,1)}, y_{A(i,2)}, y_{A(i,3)})$ match the formula's signs $(s_{A(i,1)}, s_{A(i,2)}, s_{A(i,3)})$, the conditional intensity matrix is all 0. For the other parent assignments (in which at least one variable matches), the conditional intensity matrices are $\begin{bmatrix} -1 & 1 \\ 0 & 0 \end{bmatrix}$. The initial distribution is $\begin{bmatrix} 1 & 0 \end{bmatrix}$.

Variable B_1 has parents C_1 and C_2. Variable B_i (for $1 < i < k - 1$) has parents B_{i-1} and C_{i+1}. Variable S has parents B_{k-2} and C_k. For all of these variables, the conditional intensity matrix if the two parents' values are T is $\begin{bmatrix} -1 & 1 \\ 0 & 0 \end{bmatrix}$. Otherwise, the conditional intensity matrix is all 0. All of these variables have an initial distribution of $\begin{bmatrix} 1 & 0 \end{bmatrix}$.

This reduction is polynomial in size (all numeric values are small and there are a polynomial number of variables, each with a maximum of 3 parents) and can obviously be output in polynomial time. By construction, Y_i selects at time 0 a truth value for z_i and never changes. Each C_j will then eventually change to T if and only if the clause is satisfied by the selected truth values. B_j will eventually change to be T if and only if clauses 1 through $j + 1$ are all T. S will similarly eventually change to be T if and only if all clauses are satisfied.

Because of the Markov nature of the process, for any time $t > 0$, $P_{\mathcal{G},\mathcal{Q},\pi}(S(t) = T) > 0$ if the formula is satisfiable and the same probability is 0 if the formula is not satisfiable. □

This demonstrates that determining whether a marginal is non-zero is NP-hard. By similar construction as for Bayesian networks (Koller & Friedman, 2009), this can be extended to show that absolute and relative error formulations of inference are also NP-hard.

References

Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., & Franceschinis, G. (1995). *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons.

Asmussen, S., Nerman, O., & Olsson, M. (1996). Fitting phase-type distributions via the EM algorithm. *Scandavian Journal of Statistics*, *23*, 419–441.

Baier, C., Haverkort, B. R., Hermanns, H., & Katoen, J.-P. (2000). Model checking continuous-time Markov chains by transient analysis. In *Proceedings of Computer Aided Verification*, pp. 358–372.

Baskett, F., Chandy, K. M., Muntz, R. R., & Palacios-Gomez, F. (1975). Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM*, *22*(2), 335–381.

Boutilier, C., Friedman, N., Goldszmidt, M., & Koller, D. (1996). Context-specific independence in Bayesian networks. In *Proceedings of the Twelfth International Conference on Uncertainty in Artificial Intelligence*, pp. 115–123.

Boyen, X., & Koller, D. (1998). Tractable inference for complex stochastic processes. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence*, pp. 33–42.

Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, *35*(8), 677–691.

Buchholz, P., Ciardo, G., Donatelli, S., & Kemper, P. (2000). Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models. *INFORMS Journal on Computing*, *12*(3), 203–222.

Burch, J. R., Clarke, E. M., & Long, D. E. (1991). Symbolic model checking with partitioned transition relations. In *Int. Conference on Very Large Scale Integration*, pp. 49–58. IFIP Transactions, North-Holland.

Celikkaya, E. B., Shelton, C. R., & Lam, W. (2011). Factored filtering of continuous-time systems. In *Proceedings of the Twenty-Seventh International Conference on Uncertainty in Artificial Intelligence*.

Ciardo, G., Jones, R. L., Miner, A. S., & Siminiceanu, R. (2006). Logical and stochastic modeling with SMART. *Performance Evaluation*, *63*, 578–608.

Ciardo, G., & Siminiceanu, R. (2002). Using edge-valued decision diagrams for symbolic generation of shortest paths. In *Proceedings of Formal Methods in Computer-Aided Design (FMCAD)*, LNCS 2517, pp. 256–273. Springer.

Ciardo, G., & Yu, A. J. (2005). Saturation-based symbolic reachability analysis using conjunctive and disjunctive partitioning. In *Proceedings of Correct Hardware Design and Verification Methods (CHARME)*, LNCS 3725, pp. 146–161. Springer.

Ciardo, G., Zhao, Y., & Jin, X. (2012). Ten years of saturation: a Petri net perspective. *Transactions on Petri Nets and Other Models of Concurrency*, *V*, 51–95.

Clarke, E., Fujita, M., McGeer, P. C., Yang, J. C.-Y., & Zhao, X. (1993). Multi-terminal binary decision diagrams: an efficient data structure for matrix representation. In *IWLS '93 International Workshop on Logic Synthesis*.

Codetta-Raiteri, D., & Portinale, L. (2010). Generalized continuous time Bayesian networks and their GSPN semantics. In *European Workshop on Probabilistic Graphical Models*, pp. 105–112.

Cohn, I., El-Hay, T., Friedman, N., & Kupferman, R. (2010). Mean field variational approximation for continuous-time Bayesian networks. *Journal of Machine Learning Research*, *11*(Oct), 2745–2783.

Cohn, I., El-Hay, T., Kupferman, R., & Friedman, N. (2009). Mean field variational approximation for continuous-time Bayesian networks. In *Proceedings of the Twenty-Fifth International Conference on Uncertainty in Artificial Intelligence*.

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

Deavours, D. D., Clark, G., Courtney, T., Daly, D., Derisavi, S., Doyle, J. M., Sanders, W. H., & Webster, P. G. (2002). The möbius framework and its implementation. *IEEE Transactions on Software Engineering*, *28*(10), 956–969.

Didelez, V. (2008). Graphical models for marked point processes based on local independence. *Journal of the Royal Statitical Society: Series B*, *70*(1), 245–264.

Donatelli, S. (1994). Superposed generalized stochastic Petri nets: definition and efficient solution. In *Proceedings of International Conference on Application and Theory of Petri Nets (ICATPN)*, LNCS 815, pp. 258–277. Springer.

El-Hay, T., Cohn, I., Friedman, N., & Kupferman, R. (2010). Continuous-time belief propagation. In *Proceedings of the 27th International Conference on Machine Learning*, pp. 343–350, Haifa, Israel.

El-Hay, T., Friedman, N., & Kupferman, R. (2008). Gibbs sampling in factorized continuous-time Markov processes. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, pp. 169–178.

Fan, Y., & Shelton, C. R. (2008). Sampling for approximate inference in continuous time Bayesian networks. In *Proceedings of the Tenth International Symposium on Artificial Intelligence and Mathematics*.

Fan, Y., & Shelton, C. R. (2009). Learning continuous-time social network dynamics. In *Proceedings of the Twenty-Fifth International Conference on Uncertainty in Artificial Intelligence*.

Fan, Y., Xu, J., & Shelton, C. R. (2010). Importance sampling for continuous time Bayesian networks. *Journal of Machine Learning Research*, *11*(Aug), 2115–2140.

Felsenstein, J. (1981). Evolutionary trees from DNA sequences: A maximum likelihood approach. *Journal of Molecular Evolution*, *17*, 368–376.

Fernandes, P., Plateau, B., & Stewart, W. J. (1998). Efficient descriptor-vector multiplication in stochastic automata networks. *Journal of the ACM*, *45*(3), 381–414.

Fox, B. L., & Glynn, P. W. (1988). Computing poisson probabilities. *Communications of the ACM*, *31*(4), 440–445.

Friedman, N. (1997). Learning belief networks in the presence of missing values and hidden variables. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 125–133.

Gatti, E., Luciani, D., & Stella, F. (2012). A continuous time Bayesian network model for cardiogenic heart failure. *Flexible Services and Manufacturing Journal*, *24*(4), 496–515.

Grassmann, W. K. (1977). Transient solutions in Markovian queueing systems. *Computers & Operations Research*, *4*(1), 47–53.

Gunawardana, A., Meek, C., & Xu, P. (2012). A model for temporal dependencies in event streams. In *Advances in Neural Information Processing Systems*, Vol. 24.

Heath, J., Kwiatkowska, M., Norman, G., Parker, D., & Tymchyshyn, O. (2006). Probabilistic model checking of complex biological pathways. In Priami, C. (Ed.), *Proceedings Computational Methods in Systems Biology (CMSB)*, Vol. 4210 of *Lecture Notes in Bioinformatics*, pp. 32–47. Springer Verlag.

Herbrich, R., Graepel, T., & Murphy, B. (2007). Structure from failure. In *Proceedings of the 2nd USENIX workshop on Tackling computer systems problems with machine learning techniques*, pp. 1–6. USENIX Association.

Hobolth, A., & Stone, E. A. (2009). Simulation from endpoint-conditioned continuous-time Markov chains on a finite state space, with applications to molecular evolution. *The Annals of Applied Statistics*, *3*(3), 1204–1231.

Kam, T., Villa, T., Brayton, R. K., & Sangiovanni-Vincentelli, A. (1998). Multi-valued decision diagrams: theory and applications. *Multiple-Valued Logic*, *4*(1–2), 9–62.

Kan, K. F., & Shelton, C. R. (2008). Solving structured continuous-time Markov decision processes. In *Proceedings of the Tenth International Symposium on Artificial Intelligence and Mathematics*.

Koller, D., & Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press.

Kwiatkowska, M., Norman, G., & Parker, D. (2011). PRISM 4.0: Verification of probabilistic real-time systems. In Gopalakrishnan, G., & Qadeer, S. (Eds.), *Proceedings of Computer Aided Verification*, Vol. 6806 of *LNCS*, pp. 585–591. Springer.

Kwiatkowska, M. Z., Norman, G., & Parker, D. (2004). Probabilistic symbolic model checking with PRISM: a hybrid approach. *Software Tools for Technology Transfer*, *6*(2), 128–142.

Lam, W., & Bacchus, F. (1994). Learning Bayesian belief networks: An approach based on the MDL principle. *Computational Intelligence*, *10*, 269–293.

Moler, C., & Loan, C. V. (2003). Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, *45*(1), 3–49.

Najfeld, I., & Havel, T. F. (1994). Derivatives of the matrix exponential and their computation. Tech. rep. TR-33-94, Center for Research in Computing Technology, Harvard University.

Najfeld, I., & Havel, T. F. (1995). Derivatives of the matrix exponential and their computation. *Advances in Applied Mathematics*, *16*, 321–375.

Ng, B., Pfeffer, A., & Dearden, R. (2005). Continuous time particle filtering. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pp. 1360–1365.

Nodelman, U., & Horvitz, E. (2003). Continuous time Bayesian networks for inferring users' presence and activities with extensions for modeling and evaluation. Tech. rep. MSR-TR-2003-97, Microsoft Research.

Nodelman, U., Koller, D., & Shelton, C. R. (2005). Expectation propagation for continuous time Bayesian networks. In *Proceedings of the Twenty-First International Conference on Uncertainty in Artificial Intelligence*, pp. 431–440.

Nodelman, U., Shelton, C. R., & Koller, D. (2002). Continuous time Bayesian networks. In *Proceedings of the Eighteenth International Conference on Uncertainty in Artificial Intelligence*, pp. 378–387.

Nodelman, U., Shelton, C. R., & Koller, D. (2003). Learning continuous time Bayesian networks. In *Proceedings of the Nineteenth International Conference on Uncertainty in Artificial Intelligence*, pp. 451–458.

Nodelman, U., Shelton, C. R., & Koller, D. (2005). Expectation maximization and complex duration distributions for continuous time Bayesian networks. In *Proceedings of the Twenty-First International Conference on Uncertainty in Artificial Intelligence*, pp. 421–430.

Øksendal, B. (2003). *Stochastic Differential Equations: An Introduction with Applications* (Sixth edition). Springer-Verlag.

Parikh, A. P., Gunamwardana, A., & Meek, C. (2012). Cojoint modeling of temporal dependencies in event streams. In *UAI Bayesian Modelling Applications Workshop.*

Pfeffer, A. (2009). CTPPL: A continuous time probabilistic programming language. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence*, pp. 1943–1950.

Plateau, B. (1985). On the stochastic structure of parallelism and synchronisation models for distributed algorithms. In *Proceedings of ACM SIGMETRICS*, pp. 147–153.

Portinale, L., & Codetta-Raiteri, D. (2009). Generalizing continuous time Bayesian networks with immediate nodes. In *Proceedings of the Workshop on Graph Structure for Knowledge Represetnation and Reasoning*, pp. 12–17.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). *Numerical Recipes in C* (Second edition). Cambridge University Press.

Puterman, M. L. (1994). *Markov Decision Processes*. Wiley-Interscience.

Rajaram, S., Graepel, T., & Herbrich, R. (2005). Poisson networks: A model for structured point processes. In *Proceedings of the AI STATS 2005 Workshop*.

Rao, V., & Teh, Y. W. (2011). Fast MCMC sampling for Markov jump processes and continuous time Bayesian networks. In *Proceedings of the Twenty-Seventh International Conference on Uncertainty in Artificial Intelligence*.

Rao, V., & Teh, Y. W. (2012). MCMC for continuous-time discrete-state systems. In *Advances in Neural Information Processing Systems 25*, pp. 710–718.

Rao, V., & Teh, Y. W. (2013). Fact MCMC sampling for Markov jump processes and extensions. *Journal of Machine Learning Research*, *1*, 1–26.

Roux, P., & Siminiceanu, R. (2010). Model Checking with Edge-valued Decision Diagrams. In *Proceedings of the Second NASA Formal Methods Symposium (NFM 2010), NASA/CP-2010-216215*, pp. 222–226. NASA.

Saria, S., Nodelman, U., & Koller, D. (2007). Reasoning at the right time granularity. In *Proceedings of the Twenty-third Conference on Uncertainty in AI*, pp. 421–430.

Särkkä, S. (2006). *Recursive Bayesian Inference on Stochastic Differential Equations*. Ph.D. thesis, Helsinki University of Technology.

Shimony, S. E. (1991). Explanation, irrelevance and statistical independence. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pp. 482–487.

Wan, M., Ciardo, G., & Miner, A. S. (2011). Approximate steady-state analysis of large Markov models based on the structure of their decision diagram encoding. *Performance Evaluation*, *68*, 463–486.

Warrender, C., Forrest, S., & Pearlmutter, B. (1999). Detecting intrusions using system calls: Alternative data models. In *IEEE Symposium on Security and Privacy, IEEE Computer Society*.

Weiss, J. C., Natarajan, S., & Page, D. (2012). Multiplicative forests for continuous-time processes. In *Advanced in Neural Information Processing Systems*.

Weiss, J. C., & Page, D. (2013). Forest-based point processes for event prediction from electronic health records. In *Proceedings of the European Conference in Machine Learning and Principals and Practice of Knowledge and Discovery in Databases (ECML-PKDD)*.

Williams, C. K. I. (1998). Prediction with Gaussian processes: From linear regression to linear prediction and beyond. In Jordan, M. I. (Ed.), *Learning in Graphical Models*, pp. 599–621.

Xu, J., & Shelton, C. R. (2008). Continuous time Bayesian networks for host level network intrusion detection. In *European Conference on Machine Learning*, pp. 613–627.

Xu, J., & Shelton, C. R. (2010). Intrusion detection using continuous time Bayesian networks. *Journal of Artificial Intelligence Research*, *39*, 745–774.