UCRL-TR-225272

LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Survey of Bayesian Models for Modelling of Stochastic Temporal Processes

B. Ng

October 13, 2006

This work was performed under the auspices of the U.S. Department of Energy by University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

Survey of Bayesian Models for Modelling of Stochastic Temporal Processes

Brenda Ng
Lawrence Livermore National Laboratory
Livermore, CA
`bmng@llnl.gov`

Abstract

This survey gives an overview of popular generative models used in the modelling of stochastic temporal systems. In particular, this survey is organized into two parts. The first part discusses the discrete-time representations of dynamic Bayesian networks and dynamic relational probabilistic models, while the second part discusses the continuous-time representation of continuous-time Bayesian networks.

1 Introduction

Dealing with uncertainty in complex dynamic environments is a basic challenge to the operation of real-world autonomous systems. These systems are highly complex, involving large numbers of stochastic variables and many interacting nonlinear subsystems of possibly different time scales. An autonomous system must be able to reason about the state of its components and environment in order to create informed plans of intelligent action. To achieve this goal, system analysts have to iterate between modelling and reasoning. First, one must develop a tractable model of the complex system. This model will ideally extract out irrelevant details and encode information only about the components that are central to the reasoning task. Next, given the model, one must design efficient reasoning algorithms that exploit the information presented in the model to answer a variety of queries about the system. These questions can be related to:

- State estimation and prediction: *What is the system state at the current and any other future timesteps?*
- Event reconstruction: *What past events triggered the phenomenon observed at this current timestep?*
- Anomaly/novelty detection: *When was the onset of this new system behavior?*

Depending on the performance of the inference, the model may be re-evaluated and updated to encode new information or features that emerged in the temporal process. Inference is then performed to reason about the updated model. This iterative procedure repeats until some terminal time point.

In this survey, we will examine three popular Bayesian models of temporal processes: dynamic Bayesian networks, dynamic probabilistic relational models, and continuous-time Bayesian networks. These models were developed by the artificial intelligence community, where the term *dynamic* is often used to describe a process whose state may change over time. In these models, only the state of the system variables changes, while the causal relationships between the variables are assumed constant. These causal relationships are made explicit in the graphical structure of these Bayesian models, where nodes represent variables and directed links between nodes represent the flow of influence from one variable to another. These models encode how variables evolve over time and encapsulate conditional independencies between the variables.

This survey is organized as follows. In Section 2, we introduce the notation that we will be using throughout this survey. In Section 3, we describe the basic modelling assumptions. In Section 4, we discuss discrete-time representations of temporal processes, namely the dynamic Bayesian network and its relational counterpart, the dynamic probabilistic relational model. In Section 5, we discuss the continuous-time representation of continuous-time Bayesian networks. In Section 6, we conclude with related works on social networks and other *structurally dynamic* representations that can model temporal processes, in which causal relationships between variables can change with time.

2 Notation

In this section, we introduce the notation that will be used in our discourse. We use uppercase letters to denote random variables, lowercase letters to denote their instantiations and uppercase calligraphic letters to denote the variable domains. For example, given a binary variable $Z \in \{0, 1\}$, the domain of Z is $\mathcal{Z} = \{0, 1\}$ and Z can either take on the value $z = 0$ or $z = 1$.

We use boldface when referring to a collection or set of similar items. For example, given two variables Z_1 and Z_2, the collection of the two variables is referred to as $\mathbf{Z} = \{Z_1, Z_2\}$. We also use boldface for vectors, as vectors are usually the collection of more than one element.

Subscripts and superscripts are heavily used in this survey. Time will always be indexed as a subscript. We use Z_t to denote a random variable Z at a specific time t, and $Z_{0:t}$ to denote the sequence of the Z's state from time 0 to time t. When referring to a particular variable Z in a collection of variables $\mathbf{Z}$ at a given time t, if Z occurs as the n^{th} variable in $\mathbf{Z}$, then the said variable will be referred to as $Z_{n,t}$, where the variable index n occurs as a subscript.

In general, superscripts are often reserved for indices that are specific to an inference algorithm. When this is the case, the superscripts are usually enclosed in parentheses. For example, when an inference algorithm represents the system state as a set of samples, then the index to a particular sample will occur as a superscript, e.g., $Z_t = z_t^{(i)}$ means that variable Z at time t is instantiated with the value of the i^{th} sample. In addition to having a superscript to denote the sample index, some algorithms, that employ clustering in their procedure, may also have a cluster index. To illustrate, if a set of variables $\mathbf{Z}$ belongs to a particular cluster c, then these variables at time t will be referred to as $\mathbf{Z}_t^c$.

Unless otherwise stated, the system state at time t will be denoted as $\mathbf{S}_t$. The system state $\mathbf{S}$ may consist of discrete-state random variables $\mathbf{Z}$, continuous-state random variables $\mathbf{X}$ or a mixture of the two. Observations are noisy measurements of the random variables and these observed variables are denoted by $\mathbf{Y}$.

Lastly, we use $p(\cdot)$ to denote probability densities and $P(\cdot)$ to denote probability mass functions.

3 Basic assumptions

The system state at time t is represented as $\mathbf{S}_t = \{\mathbf{Z}_t, \mathbf{X}_t\}$, where the state can consist of discrete variables $\mathbf{Z}_t$ and continuous variables $\mathbf{X}_t$. To ground our discussion, let's first assume that all variables evolve at the same fixed time granularity, so that the temporal sequence of states is equally spaced in time, i.e., $\mathbf{S}_0, \mathbf{S}_1, ..., \mathbf{S}_{t-1}, \mathbf{S}_t$. This assumption can be relaxed later on, when dealing with discrete-time processes of multiple time granularities and continuous-time processes.

The system evolves according to a first-order Markov process in which the current state captures all of the memory in the process, so that there is no additional information in the past that can be used to predict the future. This property is expressed as follows:

$$p(\mathbf{S}_t|\mathbf{S}_{0:t-1}) = p(\mathbf{S}_t|\mathbf{S}_{t-1}), \quad t = 1, 2, ... \tag{1}$$

In addition, observations depend only on the current states:

$$p(\mathbf{Y}_t|\mathbf{S}_{0:t}) = p(\mathbf{Y}_t|\mathbf{S}_t), \quad t = 1, 2, ... \tag{2}$$

Moreover, we assume that the probabilities $p(\mathbf{S}_t|\mathbf{S}_{t-1})$ and $p(\mathbf{Y}_t|\mathbf{S}_t)$ are stationary:

$$\begin{array}{lcll} p(\mathbf{S}_t|\mathbf{S}_{t-1}) & = & p(\mathbf{S}_{s+t}|\mathbf{S}_{s+t-1}), & s \geq 0 \\ p(\mathbf{Y}_t|\mathbf{S}_t) & = & p(\mathbf{Y}_{s+t}|\mathbf{S}_{s+t}), & s \geq 0 \end{array} \tag{3}$$

With these assumptions, we can characterize the process by its transition model $p(\mathbf{S}_t|\mathbf{S}_{t-1})$ and its observation model $p(\mathbf{Y}_t|\mathbf{S}_t)$.

The last assumption (Equation 3) is especially important because the invariance of $p(\mathbf{S}_t|\mathbf{S}_{t-1})$ and $p(\mathbf{Y}_t|\mathbf{S}_t)$ allows one to model the system using a static representation, such as the ones that we will be examining in this survey. However, it is also a limiting assumption, since the process and measurement mechanisms for a real-world system may drift over time, thus violating this assumption. In Section 6, we will discuss alternative representations that lift this assumption. But for now, we will assume that our temporal processes satisfy the aforementioned assumptions, as outlined in Equations 1– 3.

4 Discrete-time representation

A common approach to modelling dynamic systems is to assume that systems evolve and are measured at equally spaced time steps. As a result, most systems are modelled by a fixed time step representation and estimation on these models is done at equally spaced time steps corresponding to when state transitions or measurements may occur. This is the key idea behind representing dynamic systems using discrete-time Markov processes.

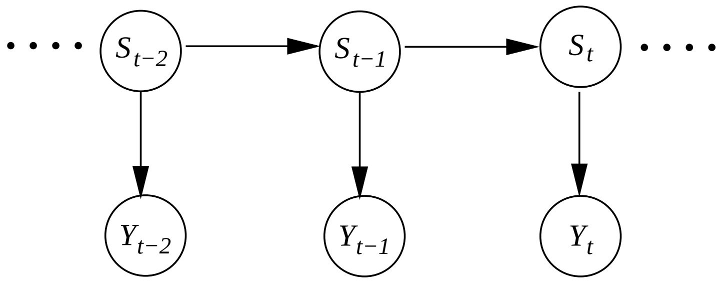

Figure 1: A simple discrete-time Markov process

Figure 1 shows a simple discrete-time Markov process with one state variable and one observed variable. At each time step t, the previous state S_{t-1} transitions to the current state S_t and a noisy measurement of S_t is generated through the observed variable Y_t. The assumptions in Equation 1 and 2 directly apply here.

As we generalize to multivariate processes, we need a way of representing these processes in a compact manner.

4.1 Dynamic Bayesian networks

In most multivariable processes, each variable is typically influenced by only a subset of the variables in the system state. As a result, one can compactly represent the transition model $p(\mathbf{S}_t|\mathbf{S}_{t-1})$ by the product of each variable's transition model:

$$p(\mathbf{S}_t|\mathbf{S}_{t-1}) = \prod_{n=1}^{N} p(S_{n,t}|\mathbf{Pa}(S_{n,t})) \tag{4}$$

where N is the number of variables in the state $\mathbf{S}_t$ and $\mathbf{Pa}(S_{n,t})$ are the *parents* of a particular variable $S_{n,t}$. A variable's parents are defined as the subset of the state variables that affects that variable. If a variable has parents, then its probability distribution will be conditional upon the values of its parents. The variables at each time step are assumed to be topologically sorted, such that $\mathbf{Pa}(S_{n,t}) \subseteq \{S_{1,t-1}, ..., S_{N,t-1}\} \cup \{S_{1,t}, ..., S_{n-1,t}\}$. In other words, a parent of the variable $S_{n,t}$ can be any variable from the previous state $\mathbf{S}_{t-1}$, or a variable in the current state $\mathbf{S}_t$ that would not induce any cyclic dependencies.

Dynamic Bayesian networks (DBNs) [Dean and Kanazawa, 1989] allow for compact representations of discrete-time Markov processes in the manner as prescribed in Equation 4. A DBN is a temporal version of a Bayesian network (BN). BNs are directed graphical models that encode conditional dependencies between state variables via graphical structure. State variables are represented as nodes and causal influences between variables are represented as arrows between nodes. Each node S_t is associated with a given conditional probability distribution $p(S_t|\mathbf{Pa}(S_t))$ that encapsulates the conditional probability of that variable given its parents $\mathbf{Pa}(S_t)$. In essence, a node is conditionally independent of its non-descendant nodes, given the values of its immediate parents.

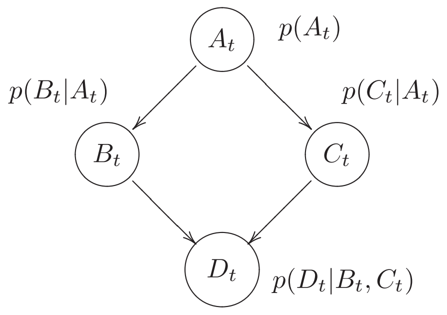

Figure 2: A simple Bayesian network

Figure 2 shows an example of a simple BN. In this BN, the variable A_t affects the variables B_t and C_t, which in turn influences the variable D_t. Given B_t and C_t, the variable D_t is independent of A_t. This idea that the conditional distribution of a variable is completely specified by its parents is important in simplifying the representation of the joint distribution of the variables. Applying the chain rule of probability, the joint distribution is given by:

$$\begin{aligned} p(\mathbf{S}_t) &= \prod_{n=1}^{N} p(S_{n,t}|S_{1,t},...,S_{n-1,t}) \triangleq \prod_{n=1}^{N} p(S_{n,t}|\mathbf{Pa}(S_{n,t})) \quad (5) \\ p(A_t, B_t, C_t, D_t) &= p(D_t|A_t, B_t, C_t) \cdot p(C_t|A_t, B_t) \cdot p(B_t|A_t) \cdot p(A_t) \\ &= p(D_t|B_t, C_t) \cdot p(C_t|A_t) \cdot p(B_t|A_t) \cdot p(A_t) \end{aligned}$$

where the last line follows from the conditional independence between the variables.

The set of nodes representing the system state at a point in time is called a *time slice*. In a BN, all nodes belong to the same time slice because a BN only models a probabilistic process at a particular point in time. To represent the temporal evolution of a dynamic process from one time point to another, a DBN has two sets of nodes, one that belongs to the current time slice and another that belongs to a previous time slice. Nodes in the previous time slice can only have parents from the same time slice while nodes in the current time slice can have parents from both time slices. To illustrate, we extend the BN from Figure 2 into a DBN and present its graphical structure in Figure 3.

From the figure, we see that the graphical structure of a DBN inherits the graphical structure of its underlying BN and includes extra structure that represents the temporal influence from the previous time slice to the current time slice. Specifically, a DBN is defined by two components: a prior Bayesian network that represents the probability distribution π_0 over the initial state, and a *2-time-slice Bayesian network* (2-TBN) that represents the transition distribution from states at time $t-1$ to states at time t. Graphically, a 2-TBN is a fragment of a Bayesian network in which the nodes belonging to the previous time slice have no parents. In our example, the 2-TBN for the DBN in Figure 3 is shown in Figure 4. In a 2-TBN, only the nodes in the current time slice are associated with conditional probability distributions. The 2-TBN represents the conditional distribution $p(\mathbf{S}_t|\mathbf{S}_{t-1})$ that encodes the transition model of the Markov process as it evolves from the previous time slice to the current time slice. In particular, $p(\mathbf{S}_t|\mathbf{S}_{t-1})$ is represented in a product form, as given in Equation 4. For any terminal time T of a process, the joint distribution of its state from time 0 to T is given by:

$$p(\mathbf{S}_0 = \mathbf{s}_0, \mathbf{S}_1 = \mathbf{s}_1, ..., \mathbf{S}_T = \mathbf{s}_T) = \pi_0(\mathbf{s}_0) \cdot \prod_{t=1}^{T} p(\mathbf{S}_t = \mathbf{s}_t|\mathbf{S}_{t-1} = \mathbf{s}_{t-1}) \quad (6)$$

In practice, $p(\cdot)$ is assumed to be a discrete probability distribution or a Gaussian distribution.

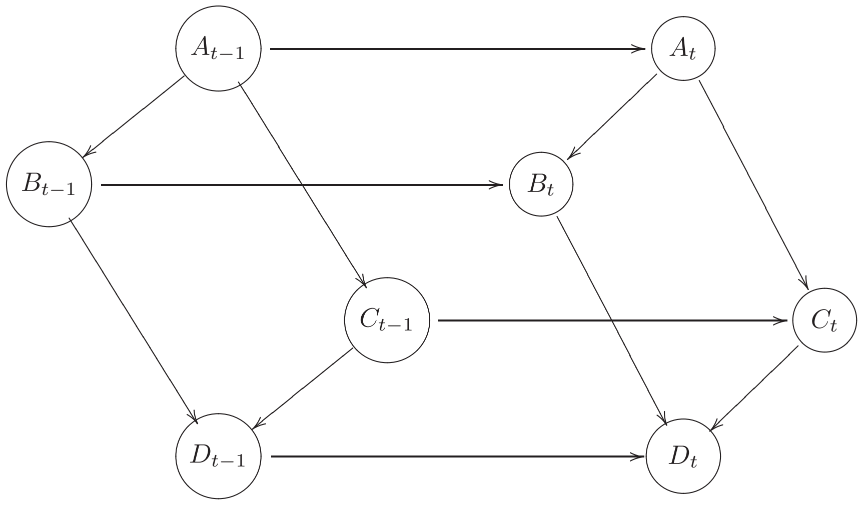

Figure 3: Graphical structure of a DBN

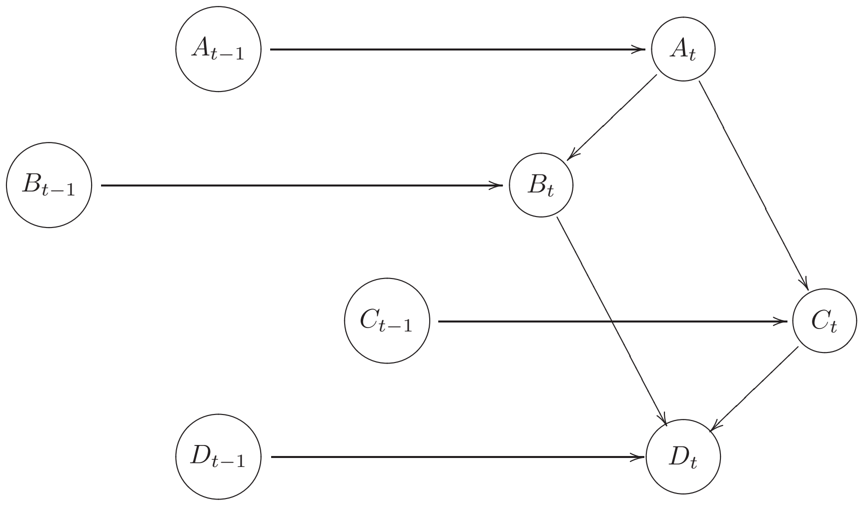

Figure 4: The 2-time-slice Bayesian network of the DBN in Figure 3

4.1.1 Empirical investigations

For simple DBNs with only a small number of variables and sparse interconnectivity between the variables, it may be possible to apply exact inference techniques—the most popular of which are variable elimination (VE) [Zhang and Poole, 1996] and junction tree propagation (JTP) [Lauritzen and Spiegelhalter, 1988; Kjaerulff, 1992]. One can apply either of the two algorithms to compute the answer to a probabilistic *query*, of the form "What is $p(\mathbf{U} = \mathbf{u}|\mathbf{V} = \mathbf{v})$?" where $\mathbf{U}$ is commonly referred to as the set of query variables and $\mathbf{V}$ is the set of observed variables.

The conceptual difference between VE and JTP is as follows: VE is a query-specific algorithm where nodes that are irrelevant to a query can be pruned away, while JTP is a more suitable algorithm for answering multiple queries because the junction tree structure allows caching of computations, which can be used to answer different queries but at the expense of higher memory requirements. A comparison of these two methods is presented in [Zhang, 1998] for the inference of Bayesian networks; the results of which also apply to DBNs. In particular, the study used the CPSC networks [Pradhan *et al.*, 1994], which are multi-level and multi-valued Bayesian networks designed for medical diagnosis, as the basis for the experiments. Both VE and JTP were tested on four CPSC-type Bayesian networks that differ in the number of nodes, the average number of parents for a node, and the average number of possible values for a node. Since both algorithms are exact methods (in that, they yield the same *correct* probabilities for a given probabilistic query), they are equally as accurate and the only difference between their performance is the runtime. As a result, the runtime is used as the performance metric in this study.

In general, it was found that VE takes less or roughly the same runtime as JTP to compute the posterior probabilities of twenty or less query variables, given a set of observations with twenty or less observation variables. Moreover, as the network complexity increases, VE is preferred over JTP because JTP simply cannot run in real time due to its immense memory requirement. Lastly, it was found that VE's runtime increases with the number of observations while JTP's runtime decreases with the number of observation variables. Although the runtimes of VE and JTP both increase with the number of query variables, it was found that VE's runtime increases at a faster rate than that of JTP. The reason is due to the JTP's junction tree structure that allows computations for previous queries to be cached and be shared among different queries. As a result, in the case of JTP, the expected time for answering the next randomly generated query decreases with the number of previous queries. Thus, as the number of observation variables and the number of query variables increase, JTP's average runtime for each separate query is faster while VE's average runtime is slower. However, the limiting factor for using JTP is that its immense memory requirement may render it infeasible for real-time inference.

For DBNs with many number of variables, approximations to exact inference are the only way to achieve real-time results. A popular approximation to JTP is the Boyen-Koller (BK) algorithm [Boyen and Kollera, 1998], where the posterior distribution is approximated as a product of marginals over C clusters,

$$p(\mathbf{S}_t|y_{1:t}) \approx \prod_{c=1}^{C} p(\mathbf{S}_t^c|y_{1:t}) \tag{7}$$

where $\mathbf{S}_t^c \subseteq \mathbf{S}_t$ is the subset of state variables that belongs to cluster c. The clusters may be disjoint or overlapping. The algorithm was tested on two real-life DBNs: the BAT network [Forbes *et al.*, 1995] and the WATER network [Jensen *et al.*, 1989], as shown in Figure 5.

Both DBNs are popular benchmarks for DBN algorithm evaluation, where the BAT model was developed for monitoring highway traffic and the WATER model was for monitoring a water purification plant. Each model was decomposed based on different clusterings and the performance metrics were accuracy and runtime. Since the findings for both models were quite similar, we present only a subset of the results for the 10-node BAT network in Table 1.

Although BK works well in practice for small-sized DBNs, the algorithm is ultimately hampered by the underlying structure of the junction tree, which often demands an exorbitant amount of memory as the complexity of the DBN scales up. To address this issue, [Murphy and Weiss, 2001] introduced the factored frontier (FF) algorithm. The FF algorithm is similar to the fully factored form of BK, which assumes a separate cluster for each variable in the DBN. For a process that spans T time steps and is modelled by a DBN that has N Q-ary state variables, where F is the maximum fan-in (the

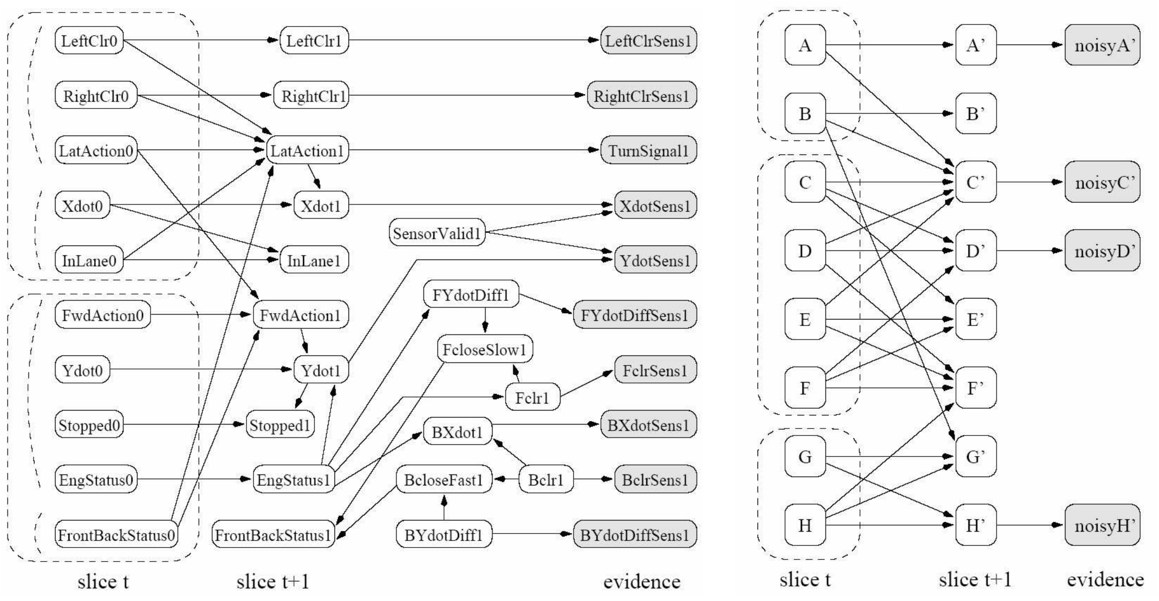

Figure 5: Two popular discrete DBNs used for benchmarking DBN algorithms: the BAT network (left) and the WATER network (right). The dotted lines correspond to the clusterings used in the BK experiments. Reproduced from [Boyen and Kollera, 1998].

Table 1: Empirical results of the Boyen-Koller algorithm on the BAT network. The clustering scheme denotes how the state variables were partitioned into groups or *clusters* used in the experiments from from [Boyen and Kollera, 1998].

Clustering scheme	Average error from true value	Speedup in runtime
5+5	0.0006	15
3+2+4+1	0.015	20
3+3+4	0.13	20

number of incoming arcs) of any node, it takes $\mathbf{O}\left(TNQ^{F+1}\right)$ time for FF to compute $P(\mathbf{S}_t|\mathbf{y}_{1:T})$, while it takes $\mathbf{O}\left(TNQ^{\sqrt{N}}\right)$ for the fully factorized version of BK.

Aside from BK and FF, another interesting approximation is presented in [Paskin, 2003], where the framework of *thin junction trees* is presented for approximate inference in the robotic task of simultaneous localization and mapping (SLAM). SLAM is an important problem in mobile robotics, because an autonomously mobile robot must map its surroundings and localize itself within its own map. In SLAM, the hidden state of the system $\mathbf{S}_t$ represents the robot's internal map, which includes the state of the robot at time t and the locations of the n_t landmarks encountered by the robot up to time t. The posterior distribution $p(\mathbf{S}_t|\mathbf{y}_{1:t})$ is approximated by a multivariate Gaussian distribution $\mathcal{N}(\mu_t, \Sigma_t)$, where μ_t represents the best estimate to the map and Σ_t is the error covariance to the estimate. For each time step, standard JTP would require $\mathbf{O}\left(Nk^2\right)$ space and $\mathbf{O}\left(Nk^3\right)$ time, where k is the width of the junction tree, defined as the size of the largest clique in the junction tree minus one. In comparison, the thin junction tree filter requires only $\mathbf{O}\left(n_t\right)$ space and $\mathbf{O}\left(n_t\right)$ time, due the constant thinning of the junction tree that reduces its width. In terms of accuracy, the thin junction tree filter achieves similar results as the Kalman filter, which is the state-of-the-art method for exact inference in SLAM, but with superior improvements in the computational resources demanded for space and time.

4.2 Hybrid-state systems

A hybrid-state system is a system that contains interacting discrete and continuous dynamics. The discrete dynamics are described by probabilistic transitions to a countable (but usually finite) set of discrete states. The continuous dynamics are described by differential or difference equations. In most cases, the discrete state dictates the set of equations under which the continuous state evolves. In return, the continuous state, upon satisfying some preset condition, may cause the discrete state to *autonomously* transition from one state to another.

In a hybrid-state system, the state is denoted by $\mathbf{S}_t = \{\mathbf{Z}_t, \mathbf{X}_t\}$, where $\mathbf{Z}_t$ denotes the discrete-state variables and $\mathbf{X}_t$ denotes the continuous-state variables. Let $\mathbf{Y}_t$ denote observed variables. To facilitate discussion, we focus on a concrete hybrid-state model, as shown in Figure 6.

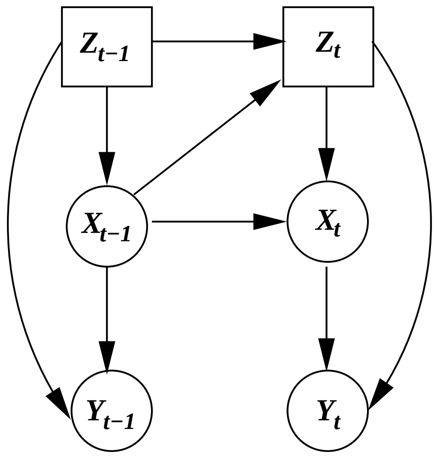

Figure 6: DBN of a hybrid-state system

For simplicity, we assume that the stochasticity in the hybrid system stems purely from additive white noise. Thus, the system is described by the following:

$$\mathbf{Z}_t \sim P(\mathbf{Z}_t|\mathbf{Z}_{t-1}, \mathbf{X}_{t-1}) \tag{8}$$
$$\mathbf{X}_t = \mathbb{F}_{\mathbf{Z}_t}(\mathbf{X}_{t-1}) + \mathbf{V}_t \tag{9}$$
$$\mathbf{Y}_t = \mathbb{H}_{\mathbf{Z}_t}(\mathbf{X}_t) + \mathbf{W}_t \tag{10}$$

where the process noise $\mathbf{V}_t \sim \mathcal{N}(0, \mathbf{Q})$ and the measurement noise $\mathbf{W}_t \sim \mathcal{N}(0, \mathbf{R})$ are mutually independent. $\mathbb{F}_{\mathbf{Z}_t}(\cdot)$ is equivalent to $\mathbb{F}(\cdot|\mathbf{Z}_t)$ but we have chosen the form of $\mathbb{F}_{\mathbf{Z}_t}(\cdot)$ to make explicit the parametrization by $\mathbf{Z}_t$. The same applies for $\mathbb{H}_{\mathbf{Z}_t}(\cdot)$. For each distinct instantiation of $\mathbf{Z}_t$, $\mathbb{F}_{\mathbf{Z}_t}(\cdot)$ and $\mathbb{H}_{\mathbf{Z}_t}(\cdot)$ comprise a unique set of equations that describes how the continuous variables $\mathbf{X}_t$ and $\mathbf{Y}_t$ evolve at each time. $\mathbb{F}_{\mathbf{Z}_t}(\cdot)$ and $\mathbb{H}_{\mathbf{Z}_t}(\cdot)$ are not assumed to be linear.

At each time, the discrete state $\mathbf{Z}_t$ is influenced by both $\mathbf{Z}_{t-1}$ and $\mathbf{X}_{t-1}$. But what does it mean for a discrete-state variable Z to be influenced by a continuous-state variable X? In many real-world applications, the continuous-state variable X_{t-1} causes an *autonomous transition* of the discrete state from Z_{t-1} to Z_t if X_{t-1} satisfies some preset condition. Let's take for example a car with automatic transmission. A car is a hybrid-state system because it has continuous-state quantities, such as speed, and discrete-state quantities, such as the transmission gear ratio. A car at rest starts from first gear and transitions to second gear when the speed exceeds 15 mph. If the speed is further increased to 27 mph, the car will shift from second gear to third gear. In more precise terms: Let X and Z represent the speed and the gear ratio respectively. For a given instantiation of Z_{t-1}, there is a *guard function* on X_{t-1} that determines a probability distribution over the discrete states for Z_t. In our car example, when the car is operating at first gear, the guard function over speed checks whether the speed exceeds 15 mph:

$$g(X_{t-1}) = \begin{cases} False & \text{if } X_{t-1} < 15 \\ True & \text{if } X_{t-1} \geq 15 \end{cases}$$

If the car is in first gear ($Z_{t-1} = 1$) and the speed does not exceed 15 mph ($g(X_{t-1}) = False$), then the gear will stay in first gear ($Z_t = 1$). This can be expressed as a probability distribution over the gear ratio Z_t:

$$\begin{aligned} P(Z_t|Z_{t-1}=1, X_{t-1}<15) &= P(Z_t|Z_{t-1}=1, g(X_{t-1})=False) && (11) \\ &= \begin{cases} 1 & \text{if } Z_t = 1 \\ 0 & \text{otherwise} \end{cases} && (12) \end{aligned}$$

In essence, the guard function g discretizes the continuous-state parent X_{t-1} into a finite number of states. As a result, the transition probability $P(Z_t|Z_{t-1}, X_{t-1})$ can now be defined as a Markov transition matrix $P(Z_t|Z_{t-1}, g(X_{t-1}))$, where Z_{t-1} and $g(X_{t-1})$ are both discrete. For each instantiation of $(Z_{t-1}, g(X_{t-1}))$, $P(Z_t|Z_{t-1}, g(X_{t-1}))$ defines a vector of transition probabilities over possible states for Z_t.

4.2.1 Empirical investigations

Due to nonlinear dynamics and complex coupling between the discrete and continuous variables, it is not always obvious how one should proceed with exact inference when it comes to hybrid DBNs. As a result, most applications resort to some form of Monte Carlo sampling techniques. The most popular of these techniques for hybrid DBN inference is the Rao-Blackwellized particle filter (RBPF) [Doucet *et al.*, 2000]. The idea of RBPF is that, given a DBN with a *tractable substructure* such that some of the variables can be marginalized out exactly from the posterior distribution, sequential Monte Carlo methods, such as the particle filter (PF) [Andrieu *et al.*, 2000], can be applied to estimate the distribution over the rest of the variables. In particular, for hybrid-state systems, only the discrete-state variables are sampled and the distribution over the continuous-state variables is computed analytically, conditional on the sampled values of the discrete-state variables. To illustrate, let's re-examine the hybrid-state system shown in Figure 6. As shown in the DBN, the transition model can be factored:

$$p(\mathbf{S}_t|\mathbf{S}_{t-1}) = p(\mathbf{X}_t|\mathbf{Z}_t, \mathbf{X}_{t-1})P(\mathbf{Z}_t|\mathbf{S}_{t-1}) \tag{13}$$

Conditional on $\mathbf{Z}_t$, the conditional posterior distribution $p(\mathbf{X}_t|\mathbf{Z}_t, \mathbf{y}_{1:t})$ can be approximated by a Gaussian distribution and the approximation can be computed analytically. (Under linear dynamics, this approximation is unnecessary because $p(\mathbf{X}_t|\mathbf{Z}_t, \mathbf{y}_{1:t})$ is exactly Gaussian.) Thus, the belief state can be expressed as follows:

$$\begin{aligned} p(\mathbf{S}_t|\mathbf{y}_{1:t}) &= p(\mathbf{X}_t|\mathbf{Z}_t, \mathbf{y}_{1:t})P(\mathbf{Z}_t|\mathbf{y}_{1:t}) && (14) \\ &\approx \mathcal{N}(\mu_t, \Sigma_t)P(\mathbf{Z}_t|\mathbf{y}_{1:t}) && (15) \end{aligned}$$

where $\mu_t \triangleq \mathbb{E}(\mathbf{X}_t|\mathbf{Z}_t, \mathbf{y}_{1:t})$ and $\Sigma_t \triangleq \mathbf{cov}(\mathbf{X}_t|\mathbf{Z}_t, \mathbf{y}_{1:t})$, which are estimated from variants of Kalman filtering techniques, such as the extended Kalman filter [Grewal and Andrews, 2001] and the unscented Kalman filter [Julier and Uhlmann, 1997; Wan and van der Merwe, 2001]. As for $P(\mathbf{Z}_t|\mathbf{y}_{1:t})$, it is estimated by a set of weighted particles drawn from a *proposal distribution*[1]:

$$P(\mathbf{Z}_t|\mathbf{y}_{1:t}) \approx \sum_{m=1}^{M} w_t^{(m)} \delta_{\mathbf{z}_t^{(m)}}(\mathbf{Z}_t) \tag{16}$$

where $\delta(\cdot)$ is the Dirac delta function.

A comprehensive empirical study of RBPF for state estimation of a hybrid-state DBN is presented in [Andersen *et al.*, 2004]. This study examined the effectiveness of RBPF for fault diagnosis on a simulated two-tank model, as shown in Figure 7. This model is often used as the benchmark model for fault diagnosis due to its simple physical interpretability and interesting nonlinear dynamics. In addition, this model encapsulates common faults:

- Measurement faults: These faults occur when a sensor fail, causing measurements to become extremely noisy.
- Burst faults: These *abrupt* faults occur when a pipe bursts, causing the pipe's resistance to change to some unknown value.
- Drift faults: These *gradual* faults occur as a result of normal wear-and-tear on a pipe, in which the pipe's resistance may drift to a non-calibrated value.

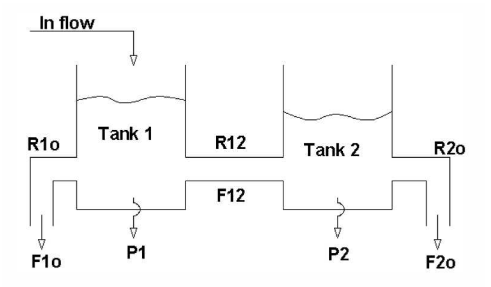

Figure 7: Schematic of the two-tank system. Reproduced from [Andersen *et al.*, 2004].

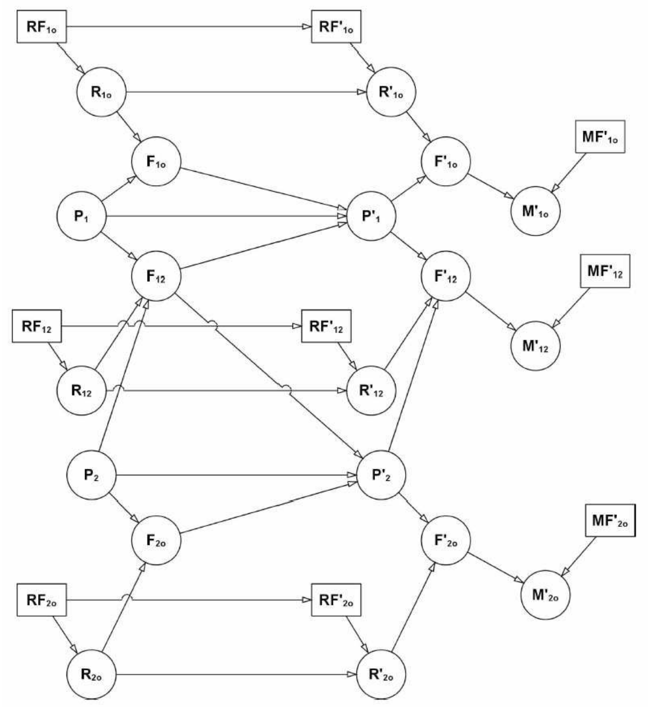

Figure 8: Hybrid-state DBN of the two-tank system. Reproduced from [Andersen *et al.*, 2004].

The DBN from Figure 8 shows the connections between the continuous system variables and the discrete fault variables (shown as RF for burst/drift faults and MF for measurement faults). To simplify the model, the pipe connecting the two tanks is constrained to not burst, which reduces the discrete state space from 32,768 states to 18,432 states. Nonetheless, the model is still too complex for exact inference methods to be feasible, so approximate inference is the only solution to state estimation.

The fault diagnosis task is to estimate the value of the discrete fault variables. Since the dynamics between the continuous variables and the discrete variables are coupled in this two-tank model, an accurate state estimate for the continuous variables would make it easier to track the discrete fault variables, and vice versa. The main goal of this empirical study is to explore the possibilities of RBPF for hybrid state estimation. In each experiment, the discrete variables $\mathbf{Z}_t$ are sampled using the standard particle filter, then the marginal probability distribution of the continuous variables, $p\left(\mathbf{X}_t|\mathbf{z}_t, \mathbf{y}_{1:t}\right)$, is estimated either by a variant of the Kalman filter, or by a specialized particle filter.

[1]Any probability distribution is a valid proposal distribution as long as it has the same support as the target distribution, which in this case is $P(\mathbf{Z}_t|\mathbf{y}_{1:t})$. In other words, if the proposal distribution is q and the target distribution is p, then $q(x) > 0$ for any x such that $p(x) > 0$. In general, the closer the proposal distribution is to the target distribution, the better the approximation of this sampling method.

In the specialized particle filter, the Kalman filter variant is used to update the proposal distribution, which is assumed to be Gaussian, so that the proposal distribution would incorporate the information from the most recent observation. This is in contrast to the standard particle filter, where the entire state $\mathbf{S}_t$ is sampled from the proposal distribution $p(\mathbf{S}_t|\mathbf{S}_{t-1})$, which is stationary and does not change with incoming observations.

The following algorithms were compared in terms of the root mean square error and the average number of wrong failure estimates:

- PF: Standard particle filter in which both $\mathbf{Z}_t$ and $\mathbf{X}_t$ are sampled
- EKF: Rao-Blackwellized particle filter using the extended Kalman filter to estimate $\mathbf{X}_t$
- UKF: Rao-Blackwellized particle filter using the unscented Kalman filter to estimate $\mathbf{X}_t$
- PFUKF: Rao-Blackwellized particle filter using a particle filter, whose proposal distribution is updated by the unscented Kalman filter, to estimate $\mathbf{X}_t$

The general recommendation from this study is that PFUKF is the most preferred method for highly nonlinear models, whose measurements are highly biased by noise and whose process and measurement models may be inaccurate. This recommendation is based on the following findings:

- PF and EKF could only estimate with reasonable accuracy the continuous variables that are directly connected to the observation variables. In contrast, UKF does not have this limitation.
- UKF and PFUKF are more affected by noise in the measurement model than in the process model, due to the fact that estimates of the state depend directly on estimates of the observed variables. In general, PFUKF outperforms UKF for high levels of measurement noise, as shown in Figure 9.
- When using an inaccurate measurement model for state estimation, PFUKF is capable of making more accurate estimates than UKF. Nonetheless, both methods outperform PF.

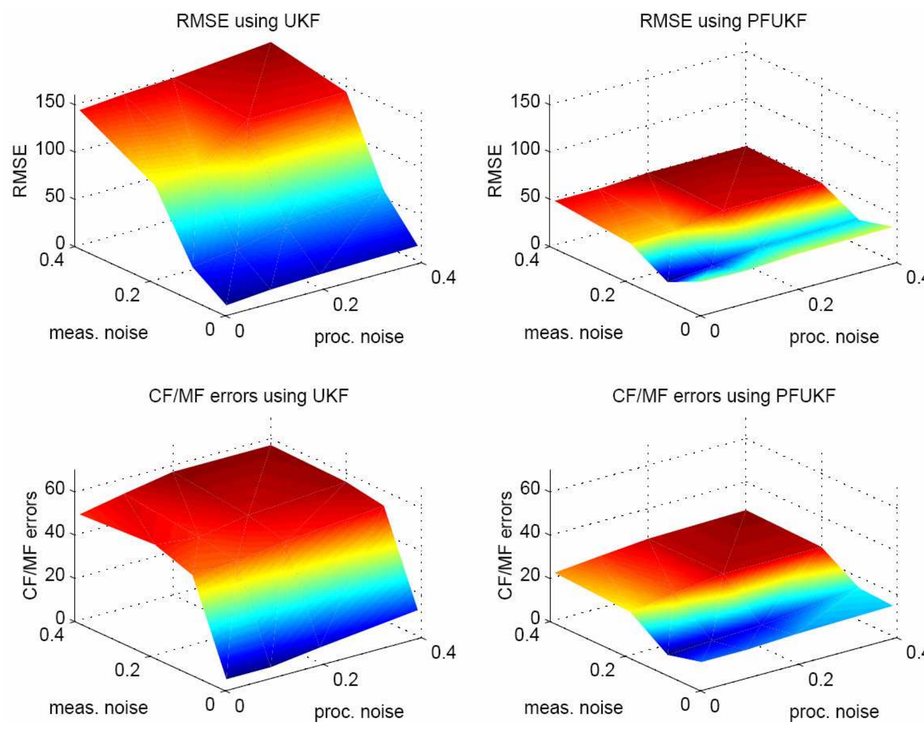

Figure 9: Surface plots of root mean square error and fault estimation error for the UKF and the PKUKF methods. Reproduced from [Andersen *et al.*, 2004].

4.3 Relational dynamic Bayesian networks

For domains with much repetitious structure, DBNs are not always the most efficient form of representation. For example, in a university setting, where there are many students and professors, each of these *entities* are usually related to one another through similar relationships. If one were to model every single aspect of student-to-student, faculty-to-student and faculty-to-faculty dynamics using a DBN, one would need to spend much time initializing the DBN fragments that represent each and every faculty and student, along with their inter-dynamics. It would be much more convenient if one could represent general principles about their relationships with statements such as "For all professors who are famous and well-funded, their students would have a higher likelihood of academic success." This generality can be achieved by extending DBNs with the power of relational logic.

Relational logic allows one to reduce a large set of propositional statements into a single one that encodes the same information with the concepts of entities and relations. With relational logic and the use of quantifiers (such as the existential quantifier $\exists$ and the universal quantifier $\forall$), one can now lift the restriction on Bayesian networks—the inability to represent general principles about multiple similar objects, which can be applied in multiple contexts.

Probabilistic relational models (PRMs) [Getoor *et al.*, 2001] are the relational analogs to Bayesian networks (BNs). While BNs define probability distributions over sets of *propositional* interpretations, in terms of instantiations to attributes or random variables, PRMs define probability distributions over sets of *relational* interpretations. In particular, one can view BNs as a special case of PRMs, whereby a BN is a stripped-down PRM with only one class of entities and no relationships between entities. To ground our discussion of dynamic PRMs, we will first explain PRMs in detail, then extend PRMs to dynamic PRMs the same way we extended BNs to DBNs. To simplify our discussion, we will only be examining models with discrete-valued variables.

A *schema* is a relational specification of a system. Given that a system is composed of basic entities that are partitioned into classes, this set of classes $\mathcal{C} = \{C_1, C_2, ..., C_k\}$ constitutes the schema for the system. Each class C in a schema is characterized by:

- a set of *propositional attributes* $\mathcal{A}(C)$ that encode the variables that comprise the class C; each proposition attribute $A.C \in \mathcal{A}(C)$ assumes values from a fixed domain $\mathcal{V}(A.C)$.

- a set of *relational attributes* $\mathcal{R}(C)$ that encode the connections between class C and other classes; each relational attribute $C.R \in \mathcal{R}(C)$ defines a mapping from the class C to the power set $2^{C'}$ of a target class $C' \in \mathcal{C}$. These relational attributes are also known as *reference slots* and can be composed together to form *slot chains* to define functions from entities to other entities to which they are indirectly related.

For example, the $University$ schema might represent the student/faculty body of a university, with the classes corresponding to different types of denizens in the university, such as tenured professors, tenure-track professors, adjunct professors, graduate students and undergraduate students. The propositional attributes of an undergraduate student might include his/her major, his/her GPA, and his/her interest in research and graduate school. The relational attributes of an undergraduate student might include the graduate student who is acting as the TA, for a course taught by a adjunct professor, who is in the same department as the undergraduate student's major.

An instance $\mathcal{I}$ of a schema is a set of entities, where each entity belongs to a class in the schema, with all propositional and relational attributes of each entity specified. Going back to our example, an instance of the $University$ schema might be a particular university, with all students and professors, their characteristics and their relationships completely specified.

Formally, a PRM specifies a probability distribution over a set of instances of a given schema. In essence, it is a template: given a set of entities, a PRM defines a probability distribution over a set of interpretations that involve these entities. But since there are infinitely many possible instances to a given schema, it is more instructive to constrain the relational model by assuming a *partial* instantiation and compute the marginal distribution over the remaining variables. This partial instantiation is an incomplete specification of an instance of a schema. One such partial instantiation is a *relational skeleton* σ, that specifies the set of entities for each class, with all relational attributes specified and all propositional attributes unspecified. In this framework, a PRM defines the probability distributions over completions $\mathcal{I}$ of any given skeleton, using the same principle of *locality of influence*—the idea that most variables are influenced by only a set of few variables.

Thus, a PRM for a schema $\mathcal{S}$ is formally defined as follows. For each class $C \in \mathcal{C}$ and each propositional attribute $A \in \mathcal{A}(C)$:

- The set of variables that influence $C.A$ is the set of parents $\mathbf{Pa}(C.A) = \{U_1, ..., U_p\}$. Each $U \in \mathbf{Pa}(C.A)$ can either be a variable within the same class (i.e., has the form $C.B$) or is an aggregation γ of variables outside the class that are referenced by a slot chain γ (i.e., has the form $\gamma(C.\gamma.B)$)
- The conditional probability distribution $P(C.A|\mathbf{Pa}(C.A))$ quantifies the causal effect that $\mathbf{Pa}(C.A)$ has on $C.A$.

Putting all this together, we can derive an equivalent expression to Equation 5, for PRMs specified by the skeleton σ:

$$P(\mathcal{I}) \quad = \quad \prod_{x\in\sigma}\prod_{A\in\mathcal{A}(x)} P(\mathcal{I}_{x.A}|\mathcal{I}_{\mathbf{Pa}(x.A)}) \tag{17}$$

where $\mathcal{I}_{x.A}$ and $\mathcal{I}_{\mathbf{Pa}(x.A)}$ denote the respective values of $x.A$ and $\mathbf{Pa}(x.A)$ in the completion $\mathcal{I}$.

Having defined PRM in detail, we are now ready to transform the static representation of PRMs into the temporal representation of *dynamic probabilistic relational models* (DPRMs) [Sanghai *et al.*, 2003]. This extension is completely analogous to the extension from BNs to DBNs. Just as a DBN is defined in terms of its prior Bayesian network (that represents the probability distribution at time 0) and a 2-time-slice Bayesian network (that describes the evolution of states between time slices), a DPRM for a relational schema $\mathcal{S}$ is defined similarly by relational structures:

- a *prior* PRM M_0 over I_0, that represents the probability distribution $\pi_0(I_0)$ over the initial instance of $\mathcal{S}$
- a *2-time-slice* PRM (2-TPRM) $M_\rightarrow$, that represents the transition distribution $P(I_t|I_{t-1})$ which describes the temporal evolution of instances from one time to the next.

A 2-TPRM is the relational analog for a 2-TBN. A 2-TPRM is a special PRM, for which each class C has been augmented with an extra relational attribute $C.previous$, with domain C. This extra attribute allows one to capture the effect of time on a particular class. Specifically, for each class C and for each propositional attribute $A \in \mathcal{A}(C)$, a 2-TPRM consists of:

- A set of parent variables $\mathbf{Pa}(C.A) = \{U_1, ..., U_p\}$, where each $U \in \mathbf{Pa}(C.A)$ can either be a variable within the same class (i.e., has the form $C.B$) or is an aggregation γ of variables outside the class that are referenced by a slot chain γ (i.e., has the form $\gamma(C.\gamma.B)$). The slot chain γ can only contain the attribute *previous* at most once. This enforces the first-order Markov property of the temporal process.
- The conditional probability distribution $P(C.A|\mathbf{Pa}(C.A))$, which quantifies the causal effect that $\mathbf{Pa}(C.A)$ has on $C.A$.

Thus, a DPRM defines a probability distribution over a temporal sequence of instances $\{\mathcal{I}_0, \mathcal{I}_1, ..., \mathcal{I}_T\}$, as follows:

$$P(\mathcal{I}_0, \mathcal{I}_1, ..., \mathcal{I}_T) \quad = \quad \pi_0(\mathcal{I}_0)\prod_{t=1}^{T} P(\mathcal{I}_t|\mathcal{I}_{t-1}) \tag{18}$$

$$\tag{19}$$

To relate DBN to DPRM: a DBN is a special case of a DPRM, whose schema contains only one class of entities and there are no relationships between entities other than the relational attribute *previous*. As a result, DRPMs are more general than DBNs, but because of this fact, DPRMs pose more difficulties in inference, as techniques for DBN inference may not scale well to DPRMs.

4.3.1 Empirical investigations

Currently, the state-of-the-art methodology for inferring about DPRMs is to apply Rao-Blackwellized particle filters [Doucet *et al.*, 2000], along with clever tricks to efficiently cache probability vectors using abstraction tree structures [Sanghai *et al.*, 2003].

In [Sanghai *et al.*, 2003], DPRMs were applied to fault detection in complex assembly plans. The study compared the performance of the particle filter (PF) [Andrieu *et al.*, 2000], the Rao-Blackwellized particle filter (RBPF), and the RBPF with abstraction trees, in monitoring plan execution. The performance metric is the Kullback-Liebler (KL) distance [Cover and Thomas, 1991], which is a distance function between two probability distributions, usually from the true or reference probability distribution to an approximate probability distribution estimated by non-exact methods. Since the problem domain is too complex for exact methods to be applied, an approximation of the KL-distance is used, which is adequate for measuring the relative differences between the approximate algorithms:

$$\hat{D}(p||\hat{p}) = -\frac{1}{S}\sum_{i=1}^{S}\log\hat{p}(x_t^{(i)}) \tag{20}$$

where $\hat{p}$ is the probability of the i^{th} sample computed from either PF or RBPF and S is taken to be 10,000 in the experiments. Figure 10 shows the comparison between PF and RBPF, in terms of the approximate KL-distance as defined in Equation 20, with varying fault probability and varying number of objects in the problem domain.

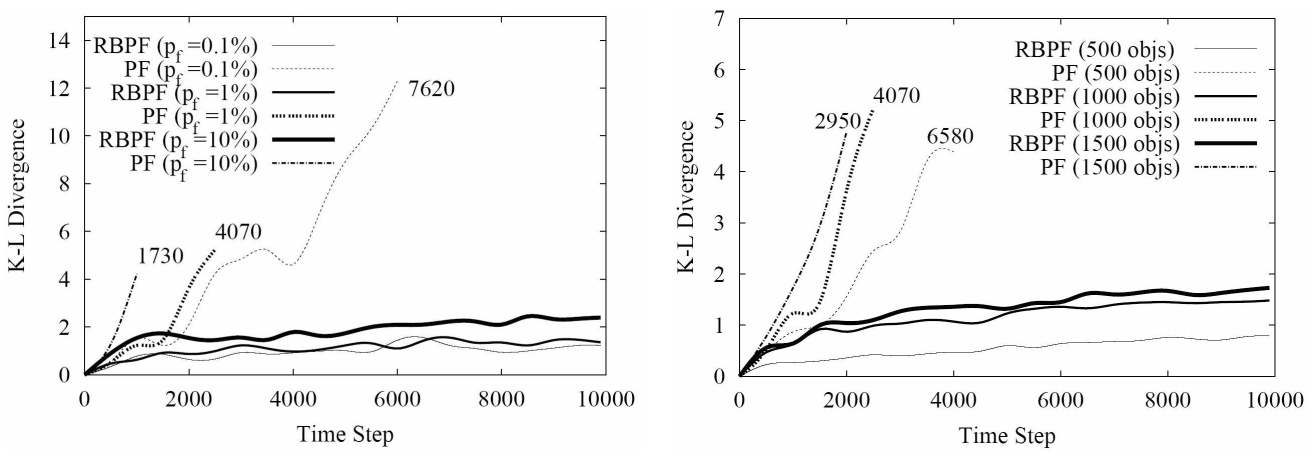

Figure 10: Comparisons of RBPF with 5000 particles and PF with 200,000 particles. The left plot shows the results for 1000 objects and varying fault probability, denoted as p_f in the graph. The right plot shows the results for varying number of objects while keeping the fault probability fixed at $p_f = 1\%$. Reproduced from [Sanghai *et al.*, 2003].

The efficiency of RBPF was further improved by the use of abstraction trees. In the study, it was found that the use of abstraction trees in RBPF reduced RBPF's runtime and memory requirement by a factor of 30 to 70. In contrast to PF, each sample from the abstraction-tree-based RBPF took 6 times longer and 11 times the memory, compared to a sample processed by PF. But since much more samples are needed by PF to reach a comparable accuracy to RBPF, the total amount of work done by RBPF still takes less time and memory.

Aside from fault detection, relational models have also been successfully applied to the modelling of online user behavior in cyberspace. In [D'Ambrosio *et al.*, 2003], PRMs were used to model the relationships between entities that interact within an internet environment. Online user modelling is interesting because it poses special challenges, such as the fact that much data violate the iid assumption on which most traditional machine learning algorithms are based. To illustrate: requests (in the form of mouse clicks) are often dependent on previous requests; sessions for a visitor are dependent on other sessions; and page types are also correlated by their link structure and by the navigation sequence of the online user.

The study by [D'Ambrosio *et al.*, 2003] examined the effectiveness of various models, from Bayesian networks to increasingly sophisticated DPRMs, in the prediction task of assessing whether or not the current mouse click is the last click in a session. The experiments were based on a sample web log of 6900 mouse clicks from a small e-commerce website and the performance metric was the AUC ROC, defined as the area under the receiver operating curve, which is a graph of the *sensitivity* (Equation 21) as a function of one minus the *specificity* (Equation 22), for a binary classification test.

$$\text{sensitivity} = \frac{\text{number of true positives}}{\text{number of true positives} + \text{number of false negatives}} \quad (21)$$

$$\text{specificity} = \frac{\text{number of true negatives}}{\text{number of true negatives} + \text{number of false positives}} \quad (22)$$

The first modelling attempt, in the form of a Bayesian network learned from the data, is shown in Figure 11. This model performed extremely poorly, resulting in only a ROC AUC of 0.274, which is even less than what could be achieved—a ROC AUC OF 0.5—by random guessing.

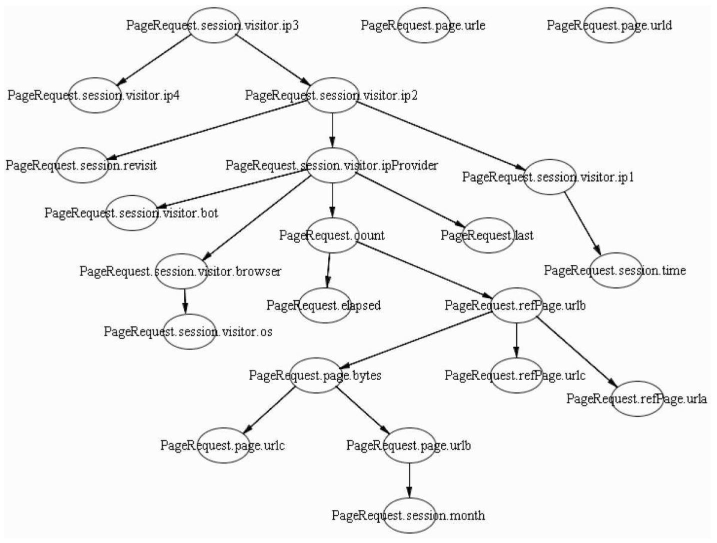
PageRequest.session.visitor.ip3
PageRequest.page.urle
PageRequest.page.urld
PageRequest.session.visitor.ip4
PageRequest.session.visitor.ip2
PageRequest.session.revisit
PageRequest.session.visitor.ipProvider
PageRequest.session.visitor.ip1
PageRequest.session.visitor.bot
PageRequest.count
PageRequest.last
PageRequest.session.visitor.browser
PageRequest.session.time
PageRequest.elapsed
PageRequest.refPage.urlb
PageRequest.session.visitor.os
PageRequest.page.bytes
PageRequest.refPage.urlc
PageRequest.refPage.urla
PageRequest.page.urlc
PageRequest.page.urlb
PageRequest.session.month

Figure 11: A static Bayesian network for predicting the last page of an internet session. Reproduced from [D'Ambrosio *et al.*, 2003].

By using DPRMs, one can incorporate temporal dependencies into the model. In particular, DPRMs allow for a more expressive representation of complex structured data and the recognition of user behavior at different time scales. The initial DPRM (not shown) achieved an ROC AUC of 0.71 and an extension of this DPRM (Figure 12) that includes a few additional links to model the dependencies between page attributes, improved the ROC AUC to 0.78.

5 Continuous-time representation

By representing a dynamic system as a discrete-time Markov process, one makes the implicit assumption that the process evolves and is observed at regularly clocked time steps. However, this assumption is unrealistic because events in the real world can occur at random times, thus violating the fixed time step assumption, or gradually, in which case it is not clear how to fix the time steps to adequately capture changes in the system.

In addition, the discrete-time representation with fixed time steps is simply inadequate for representing systems of multiple time granularities, which abound in the real world. These complex systems often consist of multiple subsystems that operate at different rates or time granularities. In order to capture all state transitions and observations in the discrete-time representation, the time steps would have to be fixed at the finest time granularity, which can be wasteful because inference would be performed on the entire system at every time step even though the system remains unchanged during

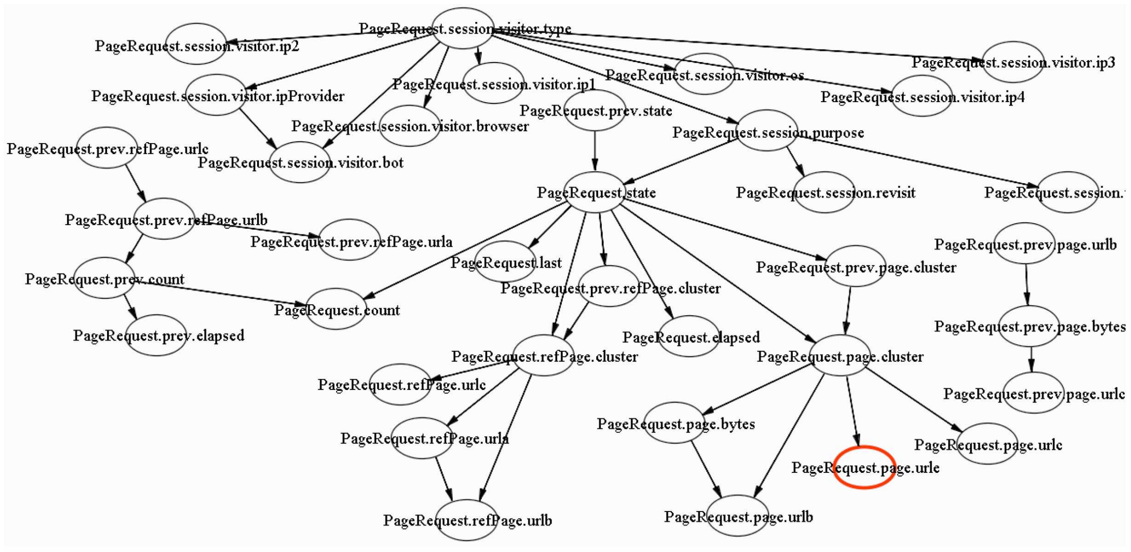
PageRequest.session.visitor.type
PageRequest.session.visitor.ip2
PageRequest.session.visitor.ip3
PageRequest.session.visitor.os
PageRequest.session.visitor.ipProvider
PageRequest.session.visitor.ip1
PageRequest.session.visitor.ip4
PageRequest.prev.state
PageRequest.session.visitor.browser
PageRequest.session.purpose
PageRequest.prev.refPage.urlc
PageRequest.session.visitor.bot
PageRequest.state
PageRequest.session.revisit
PageRequest.session.
PageRequest.prev.refPage.urlb
PageRequest.prev.refPage.urla
PageRequest.prev.page.urlb
PageRequest.last
PageRequest.prev.page.cluster
PageRequest.prev.count
PageRequest.prev.refPage.cluster
PageRequest.count
PageRequest.prev.elapsed
PageRequest.elapsed
PageRequest.prev.page.bytes
PageRequest.refPage.cluster
PageRequest.page.cluster
PageRequest.refPage.urlc
PageRequest.prev.page.urlc
PageRequest.refPage.urla
PageRequest.page.bytes
PageRequest.page.urlc
PageRequest.page.urle
PageRequest.refPage.urlb
PageRequest.page.urlb

Figure 12: An extended DPRM for predicting the last page of an internet session. Reproduced from [D'Ambrosio *et al.*, 2003].

most of these finely grained time steps. One solution would be to maintain separate discrete-time representations for each time granularity. But this only works if the time granularities are known beforehand and that subsystems evolve at fixed time steps according to their own time granularity.

An alternative approach is to avoid the issue of time granularities all together and consider time as a continuous-state random variable instead of a pre-discretized quantity. In this framework, we can represent processes whose state transitions and observations occur at random times. In essence, if we were to take a discrete-state Markov process and alter its "clock" so that time points arrive randomly, according to an exponential distribution, then the resulting process is a continuous-time Markov process.

In this section, we will discuss continuous-time representations: continuous-time counterparts to the discrete-time Markov process and DBNs. There is a large body of work on stochastic processes that provides various calculi for reasoning about continuous-time stochastic processes. For a thorough introduction to continuous-time stochastic processes and stochastic calculus, readers should refer to [Karlin and Taylor, 1975; Karatzas and Shreve, 2004; Stroock, 2003].

5.1 Continuous-time Markov processes

A continuous-time Markov process is a process that obeys the Markov assumption and treats time as a non-negative, continuous quantity. The state space of a continuous-time Markov process can either be discrete-valued, real-valued or hybrid. Depending on the nature of the state space, the process is characterized by a matrix of transition rates and/or a set of differential equations. These representations are analogs of the transition probability matrix and the difference equations that characterize discrete-time processes.

In the continuous-time paradigm, discrete-state systems are modelled by continuous-time Markov processes in the form of jump processes. A jump process is a process that makes instantaneous transitions from one state to another at random times. Specifically, a continuous-time Markov jump process is a random variable Z_t parametrized by time $t \in [0, \infty)$. Z_t starts in an initial state z_0 and remains in this state for a random amount of time before it makes a transition to a different state. The time that Z_t stays in a particular state is exponentially distributed, due to the Markovian nature of the process. A generic Markov jump process is depicted in Figure 13.

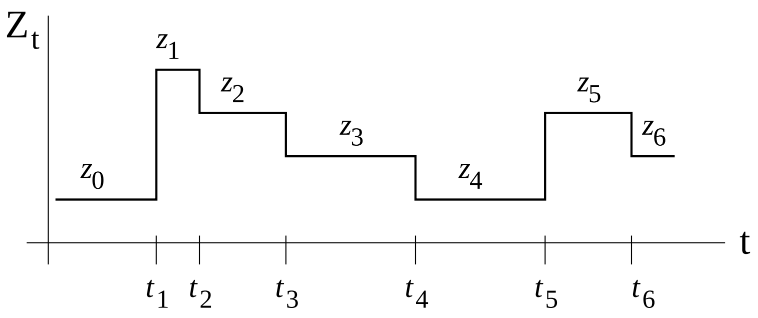

Figure 13: A generic Markov jump process Z_t

Mathematically, a Markov jump process Z_t, with finite state space $\mathcal{Z} = \{1, 2, ..., M\}$, is characterized by a M-by-M *intensity matrix*:

$$\mathbb{Q}_Z = \begin{bmatrix} -q_1 & q_{12} & \cdots & q_{1M} \\ q_{21} & -q_2 & \cdots & q_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ q_{M1} & q_{M2} & \cdots & -q_M \end{bmatrix} \tag{23}$$

where

$$q_{ij} = \lim_{\Delta t \to 0} \frac{P(Z_{t+\Delta t} = j | Z_t = i)}{\Delta t}, \quad i \neq j \tag{24}$$

$$q_i = \sum_{j:\ j \neq i} q_{ij} \tag{25}$$

in which the *transition rate* q_{ij} defines the probability per time unit that the system makes a transition from state i to state j; and q_i defines the total transition rate out of state i. Once the process enters a particular state i, the amount of time that the process stays in state i is distributed according to the exponential distribution

$$f_i(t) = q_i \exp(-q_i t). \tag{26}$$

When the process leaves state i, it enters the next state j with the transition probability:

$$P_{ij} = \begin{cases} \dfrac{q_{ij}}{q_i} & \text{if } i \neq j \\ 0 & \text{otherwise} \end{cases} \tag{27}$$

An intuitive way to interpret this representation is to assume that we have a continuous-time clock, where each tick, denoting the passage of time, is modelled by an exponential random variable. After an exponentially distributed amount of time has passed, the clock will tick and triggers the system to jump to its next state according to the transition probabilities shown in Equation 27. These probabilities are the same as the transition probabilities in a discrete-state Markov process. If instead, our clock ticks at regularly spaced time steps, then our process would reduce to a discrete-time Markov process.

This representation can be easily generalized to multivariate processes. When the discrete state **Z** consists of more than one variable Z, then we can either maintain separate jump processes for each Z, if the Zs represent independent processes; or, represent **Z** by a single jump process, whose state space is the joint space of **Z** and whose transition intensities would be given by a different intensity matrix $\mathbb{Q}_{\mathbf{Z}}$. This *joint intensity matrix* $\mathbb{Q}_{\mathbf{Z}}$ is defined similarly as $\mathbb{Q}_Z$ (Equation 23), where the intensities in $\mathbb{Q}_{\mathbf{Z}}$ are now defined over transitions from one joint state $\mathbf{z}_i$ to another joint state $\mathbf{z}_j$.

For obvious reasons, continuous-time Markov processes with real-valued state space cannot be represented as jump processes. To capture all transition intensities between states, the intensity matrix would need to be of infinite dimensions. Moreover, continuous-state systems rarely jump from one state to another. Instead, their states form a continuum and the evolution of these states are better described by a set of equations.

In discrete-time representations, continuous-state systems are modelled by a set of difference equations, of the form shown in Equations 9 and 10. In the continuous-time version, these difference equations are replaced by differential equations, which can describe the state evolution at any point in time, not just at fixed time steps. We denote the continuous-time, continuous-state process by the state vector $\mathbf{X}_t$, where $t \in [0, \infty)$. $\mathbf{X}_t$ evolves according to the following dynamics:

$$\frac{d\mathbf{X}_t}{dt} = \mathbb{F}(\mathbf{X}_t) + \mathbf{V}_t \tag{28}$$

and is observed through $\mathbf{Y}_t$, which depends on the current state:

$$\mathbf{Y}_t = \mathbb{H}(\mathbf{X}_t) + \mathbf{W}_t \tag{29}$$

$\mathbb{F}$ and $\mathbb{H}$ are functions that describe the deterministic parts of $\mathbf{X}_t$'s transition and observation models. The process and observation noises are given by $\mathbf{V}_t$ and $\mathbf{W}_t$ respectively, and their means and covariances are defined as follows:

$$\begin{array}{rclrcl} \mathbb{E}[\mathbf{V}_t] & = & \mathbf{0}, & \mathbb{E}[\mathbf{V}_t\mathbf{V}_\tau^{\mathrm{T}}] & = & \mathbf{Q}_t\ \delta(t-\tau) \\ \mathbb{E}[\mathbf{W}_t] & = & \mathbf{0}, & \mathbb{E}[\mathbf{W}_t\mathbf{W}_\tau^{\mathrm{T}}] & = & \mathbf{R}_t\ \delta(t-\tau) \\ & & & \mathbb{E}[\mathbf{V}_t\mathbf{W}_\tau^{\mathrm{T}}] & = & \mathbf{0} \end{array} \tag{30}$$

where $\mathbf{Q}_t$ and $\mathbf{R}_t$ are positive-definite covariance matrices, and $\delta(X) = \lim_{a \to 0} \delta_a(X)$ is the Dirac delta function that assigns 0 to any $x \neq 0$.

In either case of discrete or continuous state space, it is much simpler to assume that the continuous-time Markov processes are time-homogeneous, meaning that the process evolves according to the same intensity matrix or the same set of differential equations at any time. This assumption greatly simplifies the representation, because only one set of time-invariant parameters is required to characterize the process. This is a common assumption used in most representations of temporal processes, including those that are discrete-time.

5.2 Continuous-time Bayesian networks

A continuous-time Bayesian network (CTBN) [Nodelman *et al.*, 2002] is the continuous-time analog of a discrete-state DBN. In the same way that a DBN provides a factored representation of a discrete-time Markov process, a CTBN provides a compact representation for continuous-time discrete-state Markov processes.

Let $\mathbf{Z}$ denote the discrete-state variables in the system. In discrete time, the transition model $P(\mathbf{Z}_t|\mathbf{Z}_{t-1})$ is defined by the transition probability matrix over the joint states of the system. Using a discrete-state DBN, one can represent $P(\mathbf{Z}_t|\mathbf{Z}_{t-1})$ compactly in a factored form, as a product of conditional probability tables:

$$P(\mathbf{Z}_t|\mathbf{Z}_{t-1}) = \prod_n P(Z_{n,t}|\mathbf{Pa}(Z_{n,t})) \tag{31}$$

where $\mathbf{Z}_{n,t}$ is the n^{th} variable in the network and $\mathbf{Pa}(\mathbf{Z}_{n,t})$ are the parent variables to $\mathbf{Z}_{n,t}$.

In a similar manner, one can do the same with intensity matrices in a continuous-time representation. In continuous time, the joint intensity matrix $\mathbb{Q}_{\mathbf{Z}}$ parametrizes the transition model:

$$P(\mathbf{Z}_t|\mathbf{Z}_s) = \exp(\mathbb{Q}_{\mathbf{Z}}(t-s)) \tag{32}$$

Assume that each variable $Z_{n,t}$ is a continuous-time Markov jump process, characterized by the intensity matrix $\mathbb{Q}_{Z_n}$. Then one can write the continuous-time equivalent of Equation 31, as follows:

$$\begin{aligned} P(\mathbf{Z}_t|\mathbf{Z}_s) &= \exp(\mathbb{Q}_{\mathbf{Z}}(t-s)) &(33)\\ &= \exp\left(\prod_n \mathbb{Q}_{Z_n|\mathbf{Pa}(Z_n)}(t-s)\right) &(34) \end{aligned}$$

where

$$\mathbb{Q}_{\mathbf{Z}} = \prod_n \mathbb{Q}_{Z_n|\mathbf{Pa}(Z_n)} \tag{35}$$

The term $\mathbb{Q}_{Z_n|\mathbf{Pa}(Z_n)}$ is known as the *conditional intensity matrix* (CIM) for the variable Z_n. Like a conditional probability table, a CIM encodes information about how a variable evolves given the values of its parents. In particular, let Z be a M-ary random state variable and let $\mathbf{U} = \mathbf{Pa}(Z)$ be the set of parent variables that influence Z. A CIM $\mathbb{Q}_Z$ is a set of intensity matrices, where for each unique instantiation of the parent variables $\mathbf{U}$, there is a corresponding intensity matrix over the states of Z:

$$\mathbb{Q}_{Z|\mathbf{U}} = \begin{bmatrix} -q_1(\mathbf{U}) & q_{12}(\mathbf{U}) & \dots & q_{1M}(\mathbf{U}) \\ q_{21}(\mathbf{U}) & -q_2(\mathbf{U}) & \dots & q_{2M}(\mathbf{U}) \\ \vdots & \vdots & \ddots & \vdots \\ q_{M1}(\mathbf{U}) & q_{M2}(\mathbf{U}) & \dots & -q_M(\mathbf{U}) \end{bmatrix} \tag{36}$$

As shown in Equation 35, we need a product operation that combines CIMs together to form the joint intensity matrix $\mathbb{Q}_{\mathbf{Z}}$. The product operation over two CIMs is known as *amalgamation*, denoted by the symbol $(*)$. Let $\mathbb{Q}_{\mathbf{Z}_1|\mathbf{C}_1}$ and $\mathbb{Q}_{\mathbf{Z}_2|\mathbf{C}_2}$ be two CIMs and denote $\mathbf{Z} = \mathbf{Z}_1 \cup \mathbf{Z}_2$ and $\mathbf{C} = (\mathbf{C}_1 \cup \mathbf{C}_2)\backslash\mathbf{Z}$.[2] The product CIM

$$\mathbb{Q}_{\mathbf{Z}|\mathbf{C}} = \mathbb{Q}_{\mathbf{Z}_1|\mathbf{C}_1} * \mathbb{Q}_{\mathbf{Z}_2|\mathbf{C}_2} \tag{37}$$

is an intensity matrix whose:

- off-diagonal elements are zeros or intensities from $\mathbb{Q}_{\mathbf{Z}_1|\mathbf{C}_1}$ or $\mathbb{Q}_{\mathbf{Z}_2|\mathbf{C}_2}$
- diagonal elements are negative sums of the rows' off-diagonal elements, as shown in Equation 25

Adapting from [Nodelman *et al.*, 2002], the formal definition of amalgamation is as follows. Let:

- $\mathbb{Q}_{\mathbf{Z}|\mathbf{C}}(\mathbf{z}_i \to \mathbf{z}_j|\mathbf{c}_k)$ be the element in $\mathbb{Q}_{\mathbf{Z}|\mathbf{C}}$ that corresponds to the transition intensity from $\mathbf{Z} = \mathbf{z}_i$ to $\mathbf{Z} = \mathbf{z}_j$, conditional on $\mathbf{C} = \mathbf{c}_k$;
- $\mathbf{z}[\mathbf{Z}_l]$ be the projection of $\mathbf{z}$ onto the variables $\mathbf{Z}_l$, so that $\mathbf{z}[\mathbf{Z}_l]$ contains only the instantiations to $\mathbf{Z}_l$ that are consistent with $\mathbf{z}$;
- $(\mathbf{z}_i, \mathbf{c}_k)$ be the joint instantiation given by $\mathbf{Z} = \mathbf{z}_i$ and $\mathbf{C} = \mathbf{c}_k$;
- $d_H(\mathbf{z}_i, \mathbf{z}_j)$ be the Hamming distance between states $\mathbf{z}_i$ and $\mathbf{z}_j$, which represent the number of single-variable transitions that it takes for $\mathbf{Z}$ to get from $\mathbf{z}_i$ to $\mathbf{z}_j$.

Then:

$$\mathbb{Q}_{\mathbf{Z}|\mathbf{C}}(\mathbf{z}_i \to \mathbf{z}_j|\mathbf{c}_k) = \begin{cases} \mathbb{Q}_{\mathbf{Z}_1|\mathbf{C}_1}(\mathbf{z}_i[\mathbf{Z}_1] \to \mathbf{z}_j[\mathbf{Z}_1]\big|(\mathbf{z}_i, \mathbf{c}_k)[\mathbf{C}_1]) & \text{if } d_H(\mathbf{z}_i[\mathbf{Z}_1], \mathbf{s}_j[\mathbf{Z}_1]) = 1 \\ \mathbb{Q}_{\mathbf{Z}_2|\mathbf{C}_2}(\mathbf{z}_i[\mathbf{Z}_2] \to \mathbf{z}_j[\mathbf{Z}_2]\ \big|(\mathbf{z}_i, \mathbf{c}_k)[\mathbf{C}_2]) & \text{if } d_H(\mathbf{z}_i[\mathbf{Z}_2], \mathbf{s}_j[\mathbf{Z}_2]) = 1 \\ -\sum_{j:\ j \neq i} \mathbb{Q}_{\mathbf{Z}|\mathbf{C}}(\mathbf{z}_i \to \mathbf{z}_j\big|c_k) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{38}$$

In the last case, the intensity is set to 0 if there are more than one transition between $\mathbf{z}_i$ and $\mathbf{z}_j$. This is because the probability of simultaneous transitions by multiple variables in the same infinitesimal point in time is zero.

The notion of CIMs allows for factored representations of the joint intensity matrices and is central to the CTBN representation. A CTBN is defined similarly to a DBN, where there are two components:

- A prior Bayesian network that represents the initial distribution over the initial state.
- A continuous transition model that encodes graphically the factored representation of the continuous-time transition model: it is a directed, possibly cyclic graph, where each node Z_n has incoming arrows from its parent nodes $\mathbf{Pa}(Z_n)$ and each node is associated with a CIM that encodes how $\mathbf{Pa}(Z_n)$ affects the transition rate of Z_n.

[2]The $(\backslash)$ operation denotes set subtraction, where given sets A and B, an element c is in set $A\backslash B$ if and only if c is in A but not in B.

An obvious way to perform exact inference is to consider the joint system as a whole and operate on the joint intensity matrix. However, this requires computing the joint intensity matrix, which is exponential in the number of variables in the system. Thus, its sheer size prohibits practical use of exact inference for continuous-time monitoring. Although one might try to exploit the CTBN's graphical structure to decompose the task of exact inference, as in the case for DBNs, the notion of temporal dependence, or entanglement [Boyen and Kollera, 1998], between variables makes this impossible. Instead, approximate techniques, based on junction tree propagation [Lauritzen and Spiegelhalter, 1988] and expectation propagation [Minka, 2001], have been proposed in [Nodelman *et al.*, 2002; 2005] for CTBN inference.

5.2.1 Empirical investigations

Since the introduction of CTBNs as a new modelling framework, CTBNs have been applied to many interesting problem domains, including fault diagnosis, reliability analysis and activity recognition. Aside from the inference methods referenced above, there have also been attempts in adapting particle filters to CTBNs for efficient and scalable inference of these continuous-time systems.

The first of these attempts was presented in [Ng *et al.*, 2005], where the continuous-time particle filter (CTPF) was introduced and applied to the fault diagnosis of an experimental Mars rover. The fault diagnosis task was to detect wheel faults that might hamper the mobility of the rover. The rover system was modelled as a hybrid-state CTBN (Figure 14) and the CTPF algorithm was applied on the CTBN model. The state estimation results are shown in Figure 15, where each wheel's true elevation

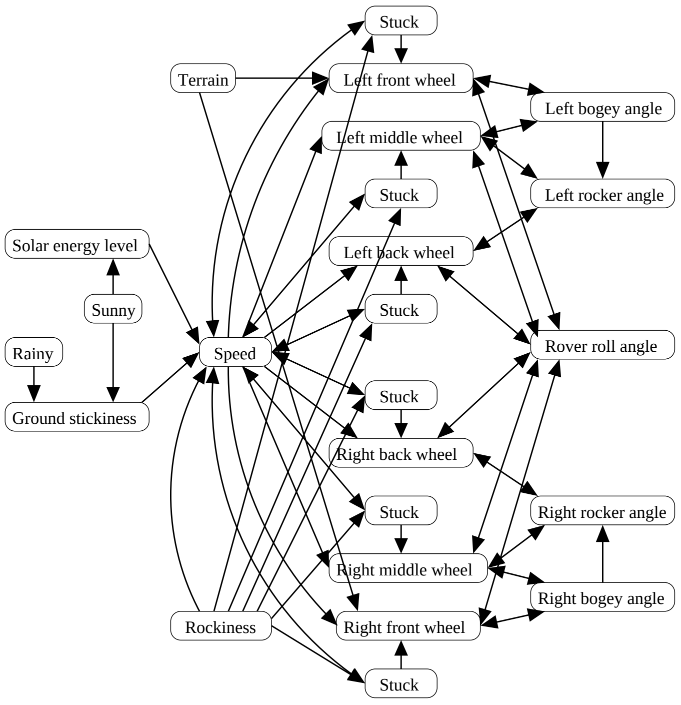

Figure 14: CTBN of the experimental Mars rover used in the [Ng *et al.*, 2005] study. Reproduced from [Ng *et al.*, 2005].

from ground level is plotted along side the estimate derived from CTPF. In general, it was found that CTPF is quite effective in tracking the rover system, despite its highly nonlinear dynamics and intermittent observations, which makes inference especially difficult because the state is observed only at very sparse time points.

While CTPF updates the state at irregular intervals, as determined by the algorithm's prediction of events, the discrete-time particle filter (DTPF) performs an update at equally spaced intervals fixed by a preset time granularity. To contrast the performance between CTPF and DTPF, the study compared the two algorithms empirically on a smaller model, the results of which are shown in Figure 16. The

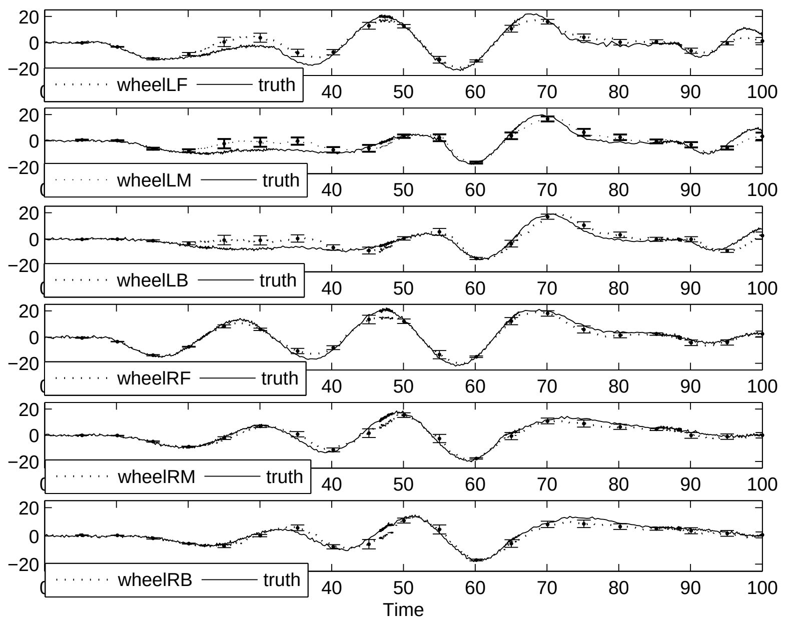

Figure 15: State estimation results from CTPF plotted against the true wheel elevations. Reproduced from [Ng *et al.*, 2005].

top two plots of Figure 16 show the results of CTPF and DTPF, where both were ran using the same parameters: the same number of samples and the same number of updates. Comparing the two plots, it is obvious that CTPF is better equipped to track oscillary behavior because it is able to allocate updates to where it is most needed, i.e., when the state is rapidly changing. But since the runtime of CTPF is higher than DTPF (due to the increased complexity from working with the continuous-time dynamics), it is only fair to compare DTPF with CTPF only if they have similar runtimes. In the bottom two plots of Figure 16, the results of DTPF with similar runtime as CTPF are shown. DTPF2 is the version of DTPF with the increased number of samples but the same number of updates as that of CTPF. In contrast, DTPF3 is the version of DTPF with the same number of samples as DTPF3 but increased number of updates. From the DTPF2 plot in Figure 16, one can see that the increase in the number of samples only drove down the variance of the state estimate, but was ineffectual in reducing the state estimate error. On the flip side, the increase in the number of updates helped the state estimate significantly, as shown in the DTPF3 plot in Figure 16. Nonetheless, CTPF still outperforms DTPF3 because its state estimate is more accurate and is with higher confidence.

Further investigation between CTBNs and DBNs is presented in [Boudali and Dugan, 2006] within the framework of reliability modelling and analysis. This study extends previous work in reliability modelling using discrete-time Bayesian networks (DTBNs), which generalize DBNs in the sense that time intervals can vary depending on the onset of events instead of fixed according to a given time granularity. The study presents a methodology for converting a dynamic fault tree into a CTBN, and outlines the advantages of CTBNs over DTBNs for reliability modelling and analysis, as shown in Table 2.

Aside from fault diagnosis and reliability analysis, CTBNs have also been applied to activity modelling and comparisons between CTBNs and DBNs have also been made on that front. In [Nodelman and Horvitz, 2003], CTBNs were applied to the prediction task of forecasting a computer user's presence and availability. The focus of the research was to model two common classes of user behavior: the presence of a user at the computer, and if the user is present, the application that the user is currently engaged in. To answer event-based and duration-related questions such as:

- When is this user next expected to be present at the computer?

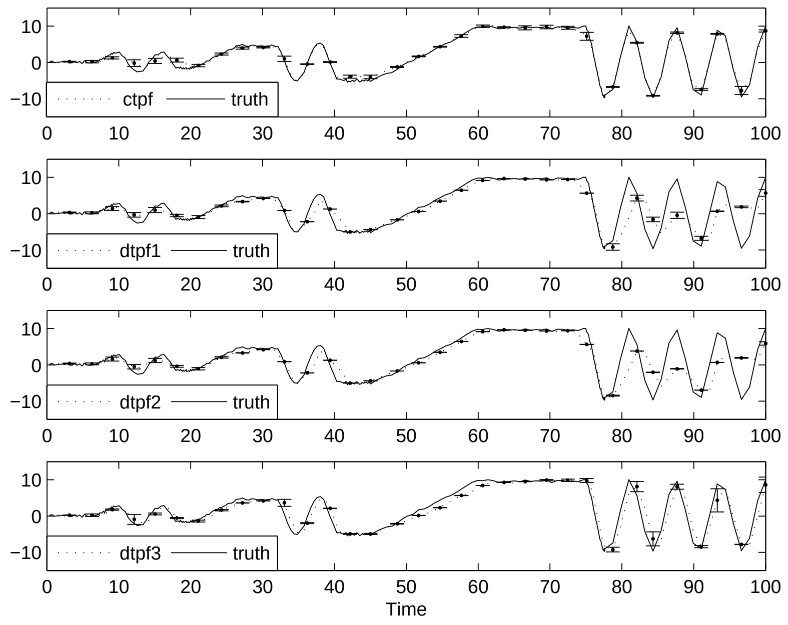

Figure 16: Comparisons between CTPF and DTPF. DTPF1 is parametrized with the same number of samples and the same number of updates as CTPF, while DTPF2 is parametrized with more samples and DTPF3 is parametrized with more updates. Reproduced from [Ng *et al.*, 2005].

Table 2: Qualitative comparison between DTBNs and CTBNs. Adapted from [Boudali and Dugan, 2006].

	Time representation
DTBN	$t \in \mathbb{N}$ (discrete)
CTBN	$t \in \mathbb{R}^+$ (continuous)
	State representation
DTBN	$\mathbf{Z}_t = failure \overset{\triangle}{=}$ The fault occurred within the time interval $((t-1)\Delta, t\Delta]$
CTBN	$\mathbf{Z}_t = failure \overset{\triangle}{=}$ The fault occurred instantaneously at time t
	Advantages
DTBN	Can be solved using standard BN inference methods
CTBN	Close-form solution for system reliability, memory savings from representing conditional probability distributions as parametric functions
	Disadvantages
DTBN	Approximate solution, high memory needs from storing conditional probability distributions as multi-dimensional tables
CTBN	No general-distribution BN inference engine

- When is the user expected to use a particular application?
- How much time is the user expected to stay at the computer?
- How much time before the user is expected to switch to a different application?

CTBNs were extended to work with a large class of phase distributions, thus generalizing CTBNs beyond the exponential distributions to handle a wider class of probabilistic queries.

To compare CTBNs with DBNs, the expected loss is used as the performance metric. The expected loss is defined as the expected value of the loss function given the distribution over when a transition will take place, and is used in place of the log likelihood, because the likelihoods of CTBNs and DBNs are not directly comparable. The study compared standard CTBNs and Erlang-based CTBNs with DBNs parametrized by different time granularities. For the Erlang-based CTBNs, the experiments used both 2 phase and 3 phase Erlang distributions. In general, it was found that the Erlang-2 CTBNs performed the best, followed by DBNs, then standard CTBNs. The Erlang-3 CTBNs were found to perform the worst. But since these models were learned by maximizing the log likelihood of data, the learned parameters were not optimized for minimizing the expected loss. As a result, one must examine other factors before deciding which model is indeed the best.

For this particular problem domain, the study found that DBNs with larger time granularities outperform those with smaller time granularities. In fact, as the time granularity approaches 0, the performance of DBN should approach that of CTBN. This is shown in Figure 17, where the expected loss is plotted against the transition time for the different models.

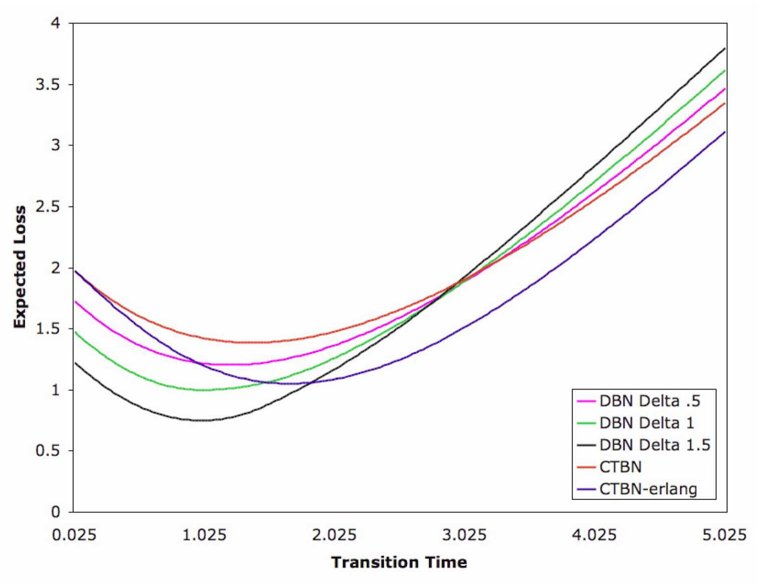

Figure 17: The expected loss as a function of the transition time for CTBNs and DBNs of different time granularities, with mean transition time for all distributions equal to 2. Reproduced from [Nodelman and Horvitz, 2003].

6 Conclusion

This survey provides an in-depth introduction to popular Bayesian models of temporal processes. These Bayesian models have been well-studied and are commonly used to model temporal processes, due to their easy interpretability and more importantly, the existing wealth of reasoning and learning tools that have already been developed for them. However, one limiting restriction of these models is that dynamics between variables are assumed unchanging over time. This is clearly not practical in real-world applications, such as the stock market, as different entities may emerge while others disappear, or as the nature or frequency of interactions between existing companies may change over time due to economic trends. Unless these Bayesian models are updated as necessary to reflect changes in the system dynamics, they would not be very useful for temporal process modelling of dynamic real-world processes.

As a result, we conclude with related works that directly model changing dynamics as part of their representation. This idea of representing changing dynamics using graphical models is directly related to the theory of dynamic graphs, in which the structure of graphs change over time. Graph theorists have studied the computational complexity of algorithms for these dynamic graphs [Demetrescu *et al.*, 2005]. In addition, others have also studied dynamic graphs in the context of social networks. In [Wasserman, 1980], strengths of relations between individuals or organizations are represented by weighted links that evolve as a Markov process. In [Sarkar and Moore, 2005], the evolution of relationships within a social network is examined using an extension of the latent space model [Hoff *et al.*, 2002] to predict whether two entities will form a connection at a future timestep, conditional on the relations that they had over past timesteps. In terms of applications, dynamic graphs have been used to model communication networks for telecommunications fraud detection [Volinsky *et al.*, 2003] and to study the spread of epidemic diseases [Newman, 2002].

Acknowledgments

This work was supported by FY'06 funding of the Predictive Knowledge Systems Strategic Initiative. In addition, much thanks to Vera Bulaevskaya, Barry Chen, Tina Eliassi-Rad, Bill Hanley, Gardar Johannesson, Phaedon-Stelios Koutsourelakis, Carol Meyers and John Nitao for their insightful comments and discussions.

References

[Andersen *et al.*, 2004] Morten Nonboe Andersen, Rasmus Ørum Andersen, and Kevin Wheeler. Filtering in hybrid dynamic bayesian networks. In *Proc. of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Montreal, Quebec, Canada, May 2004.

[Andrieu *et al.*, 2000] C. Andrieu, A. Doucet, and E. Punskaya. Sequential monte carlo methods for optimal filtering. In A. Doucet, J.F.G. de Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*, chapter 4, pages 79–93. Springer-Verlag, 2000.

[Boudali and Dugan, 2006] Hichem Boudali and Joanne Bechta Dugan. A continuous-time bayesian network reliability modeling and analysis framework. *IEEE Transactions on Reliability*, 55(1):86–97, March 2006.

[Boyen and Kollera, 1998] X. Boyen and D. Kollera. Tractable inference for complex stochastic processes. In *Proc. of the 14th Conference in Uncertainty in Artificial Intelligence (UAI)*, 1998.

[Cover and Thomas, 1991] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 1991.

[D'Ambrosio *et al.*, 2003] Bruce D'Ambrosio, Eric Altendorf, and Jane Jorgensen. Probabilistic relational models of on-line user behavior. In *Proceedings of the WebKDD-2003 Workshop on Webmining as a Premise to Effective and Intelligent Web Applications*, pages 9–16, Washington, DC, 2003.

[Dean and Kanazawa, 1989] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 1989.

[Demetrescu *et al.*, 2005] C. Demetrescu, I. Finocchi, and G. F. Italiano. Dynamic graphs. In Dinesh Mehta and Sartaj Sahni, editors, *Handbook on Data Structures and Applications*. CRC Press, 2005.

[Doucet *et al.*, 2000] A. Doucet, N. de Freitas, K. Murphy, and S. Russell. Rao-blackwellised particle filtering for dynamic Bayesian networks. In *Proc. of the 16th Conference in Uncertainty in Artificial Intelligence (UAI)*, 2000.

[Forbes *et al.*, 1995] Jeff Forbes, Tim Huang, Keiji Kanazawa, and Stuart Russell. The BATmobile: Towards a Bayesian automated taxi. In *Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1878–1885, 1995.

[Getoor *et al.*, 2001] L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In S. Dzeroski and N. Lavrac, editors, *Relational Data Mining*, chapter 13, pages 307–338. Springer-Verlag, 2001.

[Grewal and Andrews, 2001] M. S. Grewal and A. P. Andrews. *Kalman Filtering: Theory and Practice Using MATLAB*. Wiley-Interscience, 2001.

[Hoff *et al.*, 2002] P. D. Hoff, A. E. Raftery, and M. S. Handcock. Latent space approaches to social network analysis. *Journal of the American Statistical Association*, 97(460):1090–1098, 2002.

[Jensen *et al.*, 1989] F. V. Jensen, U. Kjaerulff, K. G. Olesen, and J. Pedersen. An expert system for control of waste water treatment — a pilot project. Technical report, Univ. Aalborg, Judex Datasystemer, 1989. In Danish.

[Julier and Uhlmann, 1997] S. J. Julier and J. K. Uhlmann. A new extension of the kalman filter to nonlinear systems. In *Proc. of AeroSense: The 11th International Symposium on Aeospace/Defense Sensing, Simulation and Controls*, 1997.

[Karatzas and Shreve, 2004] I. Karatzas and S. E. Shreve. *Brownian Motion and Stochastic Calculus*. Graduate Texts in Mathematics. Springer, 2004.

[Karlin and Taylor, 1975] Samuel Karlin and Howard M. Taylor. *A First Course in Stochastic Processes*. Academic Press, second edition, 1975.

[Kjaerulff, 1992] U. Kjaerulff. A computational scheme for reasoning in dynamic probabilistic networks. In *Proc. of the 8th Conference in Uncertainty in Artificial Intelligence (UAI)*, pages 121–129, 1992.

[Lauritzen and Spiegelhalter, 1988] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, 1988.

[Minka, 2001] T. P. Minka. Expectation propagation for approximate Bayesian inference. In *Proc. of the 17th Conference in Uncertainty in Artificial Intelligence (UAI)*, pages 362–369, 2001.

[Murphy and Weiss, 2001] Kevin Murphy and Yair Weiss. The factored frontier algorithm for approximate inference in dbns. In *Proc. of the 17th Conference on Uncertainty in Artificial Intelligence*, pages 378–338, San Francisco, CA, 2001. Morgan Kaufmann.

[Newman, 2002] M. E. J. Newman. Spread of epidemic disease on networks. *Physical Review E*, 66(1):016128, Jul 2002.

[Ng *et al.*, 2005] Brenda Ng, Avi Pfeffer, and Richard Dearden. Continuous time particle filtering. In *Proc. of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1360–1365, 2005.

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

[Nodelman *et al.*, 2002] U. Nodelman, C.R. Shelton, and D. Koller. Continuous time Bayesian networks. In *Proc. of the 18th Conference in Uncertainty in Artificial Intelligence (UAI)*, pages 378–387, 2002.

[Nodelman *et al.*, 2005] U. Nodelman, D. Koller, and C.R. Shelton. Expectation propagation for continuous time Bayesian networks. In *Proc. of the 21st Conference in Uncertainty in Artificial Intelligence (UAI)*, pages 431–440, Edinburgh, Scottland, UK, July 2005.

[Paskin, 2003] Mark A. Paskin. Thin junction tree filters for simultaneous localization and mapping. In Georg Gottlob and Toby Walsh, editors, *Proc. of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1157–1164, San Francisco, CA, 2003. Morgan Kaufmann Publishers.

[Pradhan *et al.*, 1994] Malcolm Pradhan, Gregory Provan, Blackford Middleton, and Max Henrion. Knowledge engineering for large belief networks. In *Proc. of the 10th Conference in Uncertainty in Artificial Intelligence (UAI)*, pages 484–490, 1994.

[Sanghai *et al.*, 2003] S. Sanghai, P. Domingos, and D. Weld. Dynamic probabilistic relational models. In *Proc. of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 992–1002, 2003.

[Sarkar and Moore, 2005] P. Sarkar and A. W. Moore. Dynamic social network analysis using latent space models. In *Advances in Neural Information Processing Systems 18 (NIPS)*, 2005.

[Stroock, 2003] D. W. Stroock. *Markov Processes from K. Itô's Perspective*. Annals of Mathematics Studies. Princeton University Press, 2003.

[Volinsky *et al.*, 2003] C. Volinsky, C. Cortes, and D. Pregibon. Computational methods for dynamic graphs. *Journal of Computational and Graphical Statistics*, 12:950–970, 2003.

[Wan and van der Merwe, 2001] E. A. Wan and R. van der Merwe. The unscented kalman filter. In S. Haykin, editor, *Kalman Filtering and Neural Networks*, chapter 7, pages 221–280. Wiley-Interscience, 2001.

[Wasserman, 1980] S. Wasserman. Analyzing social networks as stochastic processes. *Journal of The American Statistical Association*, 75(370):280–294, 1980.

[Zhang and Poole, 1996] Nevin Lianwen Zhang and David Poole. Exploiting causal independence in bayesian network inference. *Journal of Artificial Intelligence Research*, 5:301–328, 1996.

[Zhang, 1998] Nevin Lianwen Zhang. Computational properties of two exact algorithms for bayesian networks. *Applied Intelligence*, 9(2):173–183, 1998.