UNIVERSITY OF CALIFORNIA
RIVERSIDE

Continuous Time Bayesian Network Approximate Inference and Social Network Applications

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Yu Fan

December 2009

Dissertation Committee:

Dr. Christian R. Shelton, Chairperson
Dr. Gianfranco Ciardo
Dr. Neal E. Young

The Dissertation of Yu Fan is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

There are many people to whom I owe many thanks for helping me going through this long process of completing a Ph.D. First and foremost, I would like to express my gratitude to my advisor, Dr. Christian R. Shelton, for his unending support, extremely constructive feedback, excellent supervision, and all the encouragement over the last five years. Without his mentorship, this dissertation would not be possible. The experience of studying under him has been inestimable value to me.

I would also like to thank to my current and past committee members: Drs. Giangfranco Ciardo, Eamonn Keogh and Neal Young, for their support, guidance and helpful suggestions.

My deepest thanks also go to all the current and former members of Riverside Lab for Artificial Intelligence Research. Many thanks to Jing Xu for helping implement the Gibbs sampling algorithm for CTBNs in Chapter 4. Special thanks also go to former members Dr. Kin Fai Kan, Dr. Guobiao Mei; and current members Juan Casse, Busra Celikkaya, Kevin Horan, Antony Lam, William Lam, Joon Lee, and Dr. Teddy Yap for the many stimulating and enjoyable discussions. I will never forget the jokes we have shared and the food we have enjoyed over the years. I feel really lucky to be around with such a group of wonderful people.

Finally, and most importantly, I would like to thank my parents for putting their faith in me and for giving me their selfless support through all these years. Especially, I would like to give my thanks to my wife, Zhihua, who went with me through all this journey with love.

ABSTRACT OF THE DISSERTATION

Continuous Time Bayesian Network Approximate Inference and Social Network Applications

by

Yu Fan

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, December 2009
Dr. Christian R. Shelton, Chairperson

Many real world systems evolve asynchronously in continuous time, for example computer networks, sensor networks, mobile robots, and cellular metabolisms. Continuous time Bayesian Networks (CTBNs) model such stochastic systems in continuous time using graphs to represent conditional independencies among discrete-valued processes. Exact inference in a CTBN is often intractable as the state space of the dynamic system grows exponentially with the number of variables.

In this dissertation, we first focus on approximate inference in CTBNs. We present an approximate inference algorithm based on importance sampling. Unlike other approximate inference algorithms for CTBNs, our importance sampling algorithm does not depend on complex computations, since our sampling procedure only requires sampling from regular exponential distributions which can be done in constant time. We then extend it to

continuous-time particle filtering and smoothing algorithms. We also develop a Metropolis-Hastings algorithm for CTBNs using importance sampling. These algorithms can estimate the expectation of any function of a trajectory, conditioned on any evidence set containing the values of subsets of the variables over subsets of the time line.

We then apply our approximate inference algorithms to learning social network dynamics. Existing sociology models for social network dynamics require direct observation of the social networks. Furthermore, existing parameter estimation technique for these models uses forward sampling without considering the given observations, which affects the estimation accuracy. In this dissertation, we demonstrate that these models can be viewed as CTBNs. Our sampling-based approximate inference method for CTBNs can be used as the basis of an expectation-maximization procedure that achieves better accuracy in estimating the parameters of the model than the standard learning algorithm from the sociology literature. We extend the existing social network models to allow for indirect and asynchronous observations of the links. A Markov chain Monte Carlo sampling algorithm for this new model permits estimation and inference. Experiments on both synthetic data and real social network data show that our approach achieves higher estimation accuracy, and can be applied to various types of social data.

Contents

List of Figures **xi**

1 Introduction **1**

1.1 Continuous Time Bayesian Networks 1

1.2 Inference in CTBNs 7

1.3 Modeling Social Network Dynamics 10

1.4 Outline and Contribution 12

2 Continuous Time Markov Processes **15**

2.1 Representation 15

2.2 Likelihood and Sufficient Statistics 19

2.3 Query of Markov Process 20

2.4 Summary 22

3 Continuous Time Bayesian Network **23**

3.1 Structured Process Representation 23

3.2 The CTBN Model 24
3.2.1 Model Definition 25
3.2.2 Conditional Independencies 27
3.2.3 Joint Markov Process 28
3.3 Sufficient Statistics and Likelihood 30
3.4 Inference in CTBNs 31
3.4.1 Evidence and Queries 32
3.4.2 Exact Inference in CTBNs 33
3.4.3 Approximate Inference in CTBN 36
3.5 CTBN Parameter Estimation 38
3.6 Summary 39

4 Importance Sampling for CTBNs **40**

4.1 Forward Sampling 41
4.2 Importance Sampling in CTBNs 42
4.2.1 Simple Evidence 44
4.2.2 General Evidence 45
4.2.3 Predictive Lookahead 52
4.3 Particle Filtering 53
4.4 Particle Smoothing 57
4.5 Markov Chain Monte Carlo for CTBNs 60

4.5.1 Gibbs Sampling 61
4.5.2 Metropolis-Hastings Algorithm 64
4.6 Experimental Results 66
4.6.1 Networks 67
4.6.2 Evaluation Method 70
4.6.3 Experimental Results: Inference Task 71
4.6.4 Task in Parametric Estimation 80
4.7 Conclusion 82

5 Continuous-Time Social Network Dynamic Model 84

5.1 Background 85
5.2 Social Network Dynamic Models 85
5.2.1 Static Social Network Models 86
5.2.2 Discrete-time Model 86
5.2.3 Continuous-time Models 88
5.3 Network-attribute Co-evolution Model 90
5.4 Parameter Estimation 94
5.5 Summary 95

6 Learning Social Network Dynamics 97

6.1 Sampling for Learning Social Networks 98
6.1.1 Importance Sampling for Network-attribute Co-evolution Model . . . 99

6.1.2 Maximum Likelihood Estimation 100
6.2 Hidden Social Network Dynamics Model 101
6.2.1 Model Definition 102
6.2.2 Metropolis-Hastings Sampling Algorithm 104
6.2.3 Parameter Estimation 107
6.3 Experimental Results 108
6.3.1 Network-attribute Co-evolution Model 109
6.3.2 Hidden Social Network Dynamics Model 113
6.4 Conclusion 121

7 Conclusion **122**

Bibliography **125**

A Background Materials **129**

A.1 Markov Chain Monte Carlo 129
A.1.1 Markov Chain 130
A.1.2 MCMC Sampler 131
A.1.3 Gibbs Sampling Algorithm 133
A.1.4 Metropolis-Hastings Sampling Algorithm 133
A.2 Expectation Maximization 136

List of Figures

1.1 A DBN model of friendship and phone call 3

1.2 A DBN for a different time interval 5

1.3 A CTBN model of friendship and phone call 7

3.1 CTBN Example: Weight Control Effect 25

4.1 Forward sampling semantics for a CTBN 42

4.2 Example of calculating weight contribution 49

4.3 Importance sampling for CTBNs 51

4.4 Particle Filtering for CTBNs 55

4.5 Particle Smoothing for CTBNs 59

4.6 Drug Effect Network 68

4.7 British Household Panel Survey Network 69

4.8 Accuracy of importance sampling algorithms 72

4.9 Comparison of particle filtering, smoothing and importance sampling 73

4.10 Comparison to expectation propagation: Drug Network 75

4.11 Comparison to Gibbs sampling: drug network 77

4.12 Comparison to Gibbs sampling: BHPS network 78

4.13 Comparison to Gibbs sampling: chain network 79

4.14 Learning results for drug effect network 81

6.1 Log-likelihood of testing data 111

6.2 Estimated parameters for the 50 girls dataset. 112

6.3 Estimated parameters for the Enron dataset 115

6.4 Enron adjacency matrix at different times. 116

6.5 Parameters of utility function for reality mining dataset 118

6.6 Conditional intensity matrix of reality mining dataset 119

6.7 Reality mining adjacency matrix at different times. 120

Chapter 1

Introduction

Many real world applications, such as computer networks, sensor networks, social networks, mobile robots, and cellular metabolisms, involve highly complex dynamic systems. For example, a mobile robot keeps receiving signals from its sensors reflecting the changing of its environment as it moves around. Or in a computer network, each computer receives and sends hundreds of different types of packets every second. These applications usually contain a large number of stochastic variables, which evolve asynchronously in continuous time. Modeling, learning and reasoning about these complex dynamic systems is an important task and a great challenge.

1.1 Continuous Time Bayesian Networks

In all the above applications, one central task is to understand how variables of the system change and to predict when the change happens. That is, we would like to answer questions

about probability distributions of the system over time. For instance, we might want to know how likely two persons will become friends in the next six months. Since all variables evolve in continuous time, one natural approach is to use continuous-time models. Markov processes are commonly used for systems with a finite number of states. To model a multi-variable system, we need to combine all variables of the system into a single process variable. For example, suppose we want to model the friendship between two persons and their phone call communications. We need a binary variable X_{ij} to represent whether a person i is a friend of a person j, and another binary variable C_{ij} to represent whether i is calling j. To consider these two factors using a single Markov process, we need to combine these two variables together, which needs four states, each of which corresponds to a possible state combination of X_{ij} and C_{ij}. If we also include the information from j to i, i.e., X_{ji} and C_{ji}, we will need 16 states for this Markov process, each of which represents a possible combination of these four binary variables. If we want to analyze these types of dynamics with more people, the number of states of the Markov process increases exponentially.

Such growth of the state space of the Markov process makes it very difficult to represent large systems and calculate probability distributions for them. One solution is to use a structured representation to factorize the state space according to the dependencies of the variables. Bayesian networks [Pearl, 1988] are a standard method for modeling static systems, and dynamic Bayesian networks (DBNs) [Dean and Kanazawa, 1989] are commonly used for dynamic systems. A DBN describes the dynamic system as a time-sliced model by measuring the evolution of the system with a (usually fixed) time interval Δt. The transition

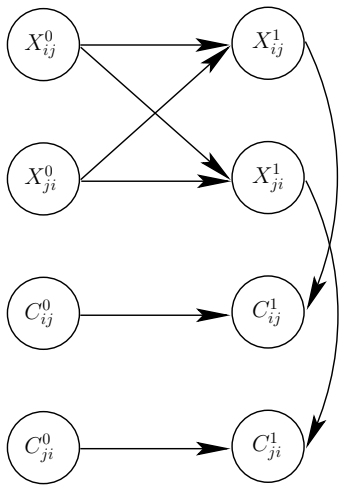


Figure 1.1: A DBN model of friendship and phone call

probabilities from states at time t to states at time $t + \Delta t$ are represented by a Bayesian network. For example, we can model the dynamics of the friendship and phone calls between i and j using a DBN shown in Figure 1.1. Variables with a superscript 0 represent variables at time step t and those with a superscript 1 represent variables at time step $t + \Delta t$. The dynamics of each variable only depend on some variables at time t and $t + \Delta t$. For example, the transition probability of X_{ij}^1 depends on X_{ij}^0, X_{ji}^0, while C_{ij}^1 depends on X_{ij}^1 and C_{ij}^0. Using the dependencies among variables, the transition probabilities between two time slices t and $t + \Delta t$ are decomposed into each variables. Thus, large dynamic systems can be represented efficiently using DBNs.

DBNs work well for systems that are observed at regular time steps. However, discretizing time line has several limitations. First, we usually choose a fixed time interval, Δt. In many real world systems, variables evolve at different time granularities. Some variables may evolve very fast whereas some evolve very slowly. Choosing an appropriate time in-

terval is a difficult task. A larger Δt may result in an inaccurate model while a smaller Δt may cause the model to be inefficient. For instance, in the friendship and phone call system in Figure 1.1, two persons may call each other four or five times every week. To accurately represent this, we have to choose the finest granularity, which is computationally inefficient as the friendship between the two persons is not expected to change for several months. If we choose a larger time interval, the resulting model is insufficient to represent the phone-call events that happen during time intervals, which makes the transition model inaccurate. Moreover, variables in a system may evolve in irregular steps. Take the phone-call event as an example. During one month, we may observe many events in the first two days and observe nothing in the rest of the month. However, once we choose the time interval, we have to propagate the distribution in each time step even when no events are observed.

Second, the dependencies of the transition model are unstable with respect to Δt. That is, different choices of Δt may result in different network structures between t and $t + \Delta t$. For example, we can unroll the DBN in Figure 1.1 for 2 time steps, which results in Figure 1.2(a). Figure 1.2(a) describes the dependencies among variables at time steps t, $t + \Delta t$ and $t + 2\Delta t$. From the figure, we can see that C_{ij}^2 also depends on X_{ij}^0, since the probability influence of X_{ij}^0 can flow from X_{ij}^1 and C_{ij}^1 to C_{ij}^2. Thus, if we choose a new interval $\Delta t' = 2\Delta t$, we will obtain a different DBN as shown in Figure 1.2(b).

Finally, DBNs (and discrete-time Markov processes in general) do not necessarily correspond to processes that are Markovian outside of the sampled instants of time. Therefore, there may not be any simple extension of a DBN to the times between the sampled instants.

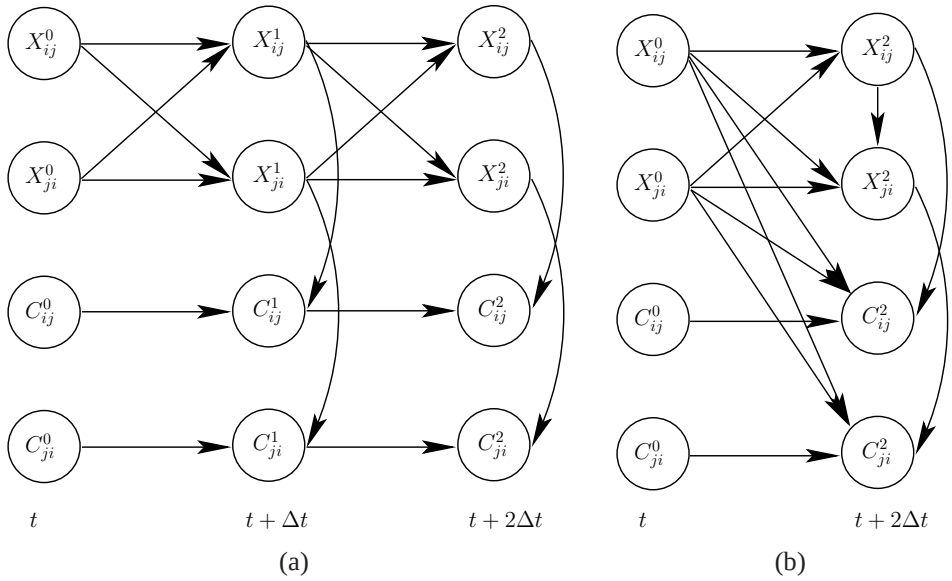


Figure 1.2: A DBN for a different time interval. (a) Unrolled DBN of friendship and phone call for 2 time steps. (b) DBN structure for a different time interval.

For example, the transition probabilities of a Markov chain with n states can be represented using an $n \times n$ matrix P, where each element P_{ij} in P describes the probability of transitioning from state i to state j in each time step from t to $t + \Delta t$. If we further want to include the Markov behaviors in the middle of each time interval $t + \frac{1}{2}\Delta t$, we need to use another transition matrix P' for the new model. Thus, the transition probabilities from t to $t + \Delta t$ under the new model can be calculated as $P' \times P'$. To preserve the Markov property in the old Markov chain, P' should satisfy $P' \times P' = P$. However, there is not always a real-valued solution for such P'.

Since discrete-time models such as DBNs have some limitations and single Markov processes suffer from the state space explosion as the number of variables increases, an ideal

solution is to use a continuous-time model that can provide a structured representation to decompose the state space.

Recently, Nodelman et al. [2002] presented *continuous time Bayesian networks* (CTBNs), which provide a structural representation of continuous-time, finite-state, Markov processes. A CTBN factorizes a multi-variable Markov process into local variables using a graphical representation. Each variable is modeled as an inhomogeneous Markov process whose dynamics depend on the current value of a set of variables in the system. For example, the friendship and phone-call model can be described using the CTBN shown in Figure 1.3. Since we use a continuous-time model, only four variables are needed, each of which is an inhomogeneous Markov process. The dynamics of each variable depend on the current instantiation of a set of other variables in the model. These dependencies are represented using the arcs in the graph. For instance, the dynamics of C_{ij} depend on the current value of X_{ij}. By doing this, CTBNs explicitly represent the temporal dynamics in continuous time and exploit the dependencies among stochastic variables using a structured representation. Thus, exponential explosion in the representation is avoided, and queries can be answered using distributions over continuous time. Because of these advantages, CTBNs have been applied to various real world systems, including human-computer interactions [Nodelman and Horvitz, 2003], server farm failures [Herbrich et al., 2007], robot monitoring [Ng et al., 2005], network intrusion detection [Xu and Shelton, 2008], and social network analysis [Fan and Shelton, 2009]. Kan and Shelton [2008] exploited the CTBN representa-

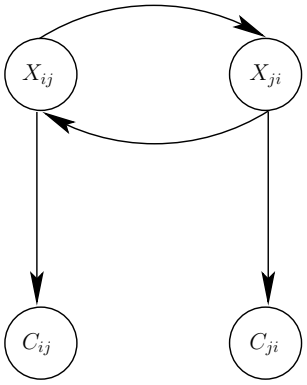


Figure 1.3: A CTBN model of friendship and phone call

tion and presented an approach to solve structured continuous-time Markov decision processes.

1.2 Inference in CTBNs

In CTBNs, a trajectory (or sample) consists of the starting values for the system along with the (real-valued) times at which the variables change, and their corresponding new values. A partial trajectory is a trajectory in which some values or transitions are missing for some variables during some time intervals. An observation (or evidence) in a CTBN is usually a partial trajectory. Inference for CTBNs is the task of estimating distributions given some observations. Inference is very important as it not only helps us answer queries about distributions, but it is also involved in parameter estimation when the observation data are incomplete. Performing exact inference in a CTBN requires combining all the variables in the CTBN into a single Markov process. This method, as discussed previously, suffers from the exponential explosion of the state space. Thus, many applications of CTBNs require an approximate in-

ference method. One method based on expectation propagation [Minka, 2001] was presented in Nodelman et al. [2005a]. Saria et al. [2007] extended it to full belief propagation and provided a method to adapt the approximation quality. Cohn et al. [2009] provided a method based on mean field variational approximation. All these methods depend on complex numerical computations, are hard to implement, and are not guaranteed to converge to the true value in the limit of infinite computing time.

Another approximate inference method is to use a sampling-based algorithm. In CTBNs, a sample is a trajectory that is consistent with the evidence. Given the evidence, a sampling-based algorithm generates a set of trajectories by simulating the dynamic system according to the evidence. Queries about CTBNs are answered using the sample trajectories. For example, if we want to know the probability that a variable $X = x$ at time t, we check the value of X at t for each sampled trajectory and calculate the ratio of the number of samples for which $X = x$ to the total number of samples as the approximated probability. Sampling has the advantage of being an anytime algorithm. That is, we can stop at any time during the computation and obtain an answer. Furthermore, in the limit of infinite samples (computation time), it converges to the true answer. Another advantage of sampling-based algorithm is that it is easy to handle any kind of query. Queries such as "the expected value of X at the time when Y transitions to value y_1 for the second time" can be calculated directly using samples, which is difficult using the other approximate inference methods mentioned above.

Sampling from dynamic systems is not new. However, most of the previous work has been in the area of discrete-time systems. Continuous-time systems pose different problems.

Any evidence containing a record of the change in a variable has a zero probability under the model. Therefore rejection sampling and straightforward likelihood weighting are generally not viable methods. Ng et al. [2005] developed a continuous-time particle filtering algorithm. However, it only handles point evidence on binary and ternary discrete variables using rejection sampling, and focuses primarily on the incorporation of evidence from the continuous-state part of the system. Recently, El-Hay et al. [2008] provided another sampling algorithm for CTBNs using Gibbs sampling. The Gibbs sampling algorithm can handle any type of evidence, and it provides an approach to sample from the exact posterior distribution given the evidence. However, the posterior distribution can be any arbitrary function. Thus, in order to sample exactly from it, binary search has to be applied and the posterior distribution has to be evaluated repeatedly, which may affect the efficiency of the algorithm.

In this dissertation, we provide another sampling-based algorithm for CTBNs using importance sampling. Our importance sampling algorithm generates weighted samples by naturally simulating the CTBN we are reasoning about. It only forces the behavior of some variables according to the upcoming evidence and calculates the corresponding weight contribution. Queries are answered using these weighted samples. Using the importance sampling algorithm, we can answer any type of query given any type of evidence. We also extend the algorithm to particle filtering and smoothing algorithms. A Metropolis-Hastings algorithm for CTBNs can also be derived based on our algorithm. The importance sampling algorithm does not depend on complex numeric computations and is easy to implement. It can be applied to many applications where the dependencies among variables are complicated and

other inference algorithms are difficult to implement. One example of such applications is social networks.

1.3 Modeling Social Network Dynamics

Social networks are a very important type of continuous-time dynamic system in our daily life. They represent the relationships (such as friendship or co-authorship) among actors (such as individuals or companies). They are highly dynamic and naturally evolve in continuous time. Understanding the dynamics of the social network allows us to predict, evaluate and control social processes more accurately. For example, it can help us control the spread of a disease or predict the reactions of terrorists.

Social networks have been studied for decades. However, the majority of the existing studies model social networks use static or discrete time models. Static models such as the p_1 model [Holland and Leinhardt, 1981], the Exponential Random Graph Model (ERGM, or p^* model) [Anderson et al., 1999], and the latent space model [Hoff et al., 2002] only focus on the static properties of social networks and usually require the network to be fully observable. Dynamic properties of social networks cannot be reflected using these models. Other models such as those of Sarkar and Moore [2005] and Guo et al. [2007] assume social networks evolve in discrete time steps and study the dynamics of social networks using discrete time models. However, actors in social networks behave very differently. Events in social networks often happen at an irregular pace. For example, a person may receive a lot of Face-

book posts on the person's birthday and only receive one or two in the next month. As we mentioned in Section 1.1, choosing the correct time interval is a great challenge.

As social networks evolve asynchronously, a continuous-time model can provide more flexibility and higher fidelity in modeling such networks. Recently, Snijders [2005] provided an actor-oriented model which considers the whole social network evolution as a continuous-time Markov process. The evolution of the network is modeled as actors making decisions to add or remove a link to maximize a utility function. Thus, each link in the network is modeled as an inhomogeneous Markov process, whose intensity depends on the structure of the network and changes over time. Dynamics of the social networks in this model purely depend on the topology of the networks. Usually, the characteristic attributes of the actors and the networks structure (both time-variant) depend on each other. For example, people who have the same interests are likely to become friends and friends are likely to influence each other's interests. Such effects should be considered when modeling social networks. Snijders et al. [2007] extended the actor-oriented model to the network-attribute co-evolution model which added effects between the network structure and the actors attributes. These two social network dynamic models allow us to model the dynamics of social networks in continuous time and allow the social network dynamics to depend on the entire structure of the network and actors attributes. However, there are two aspects upon which we can improve.

First, the parametric estimation in Snijders [2005] and Snijders et al. [2007] was implemented using a forward sampling method. However, this method can only handle data that

completely specifies all variables at discrete time instants. The samples generated by their forward sampling algorithm are not entirely consistent with the observations and the information provided by the observations is only partially used during the learning procedure. Second, parametric estimation for the network-attribute co-evolution model requires observations of the network as it evolves (at least three snapshots at different times). Usually, direct observation of a social network is very expensive. The scarceness of the data could result in an inaccurate estimation of the model.

In this dissertation, we address these two issues by demonstrating the relationship between social network dynamic models and CTBNs. Our importance sampling algorithm for CTBNs can be applied to social network dynamic models with small modifications. Using importance sampling, we can handle asynchronous, partial and duration observations, and improve the previous parametric estimation methods for social network dynamics. We address the second issue by noticing the fact that other observations, such as communication events among people (emails and instant messages), can reflect people's relationships indirectly. More importantly, they are easier to collect. We develop a *hidden social network dynamics model*, in which indirect observations such as emails events among people can be utilized.

1.4 Outline and Contribution

In this dissertation, we explore sampling-based approximate inference algorithms for CTBNs using importance sampling. We then discuss how to apply these algorithms to real world applications, such as social network analysis. The remainder of the dissertation is structured as following:

In Chapter 2, we review the background of continuous time Markov processes, which is the basis of both CTBNs and continuous-time social network dynamics models.

In Chapter 3, we review the background of CTBNs, including the model definition, exact inference and approximate inference in CTBNs, and parametric estimation using complete and partial observations.

In Chapter 4, we discuss sampling-based approximate inference algorithms for CTBNs using importance sampling. We explain our importance sampling algorithm for CTBNs, and then extend the algorithm to particle filtering and smoothing algorithms. We also demonstrate a Metropolis-Hastings algorithm for CTBNs.

In Chapter 5, we review continuous-time social network dynamics models. We introduce the actor-based models, the network-attribute co-evolution models and the method of moments algorithm for parameter estimation in these models.

In Chapter 6 we show how to apply CTBNs and our importance sampling algorithms to continuous-time social network dynamic models. We also provide our hidden social network

dynamic model and explain a parametric estimation algorithm using our Metropolis-Hastings sampling algorithm.

Chapter 7 concludes this dissertation with a summary and discussion.

The contributions of this dissertation are represented in Chapter 4 and Chapter 6. More specifically, the contributions of this dissertation are:

- We develop an importance sampling algorithm for CTBNs. The algorithm can handle any type of observation and does not depend on complex numeric computations. It can be easily extended to continuous time systems other than CTBNs.

- We extend the importance sampling algorithm to particle filtering and smoothing algorithms.

- We present a Metropolis-Hastings algorithm for CTBN based on the importance sampling algorithm.

- We introduce the CTBN model and our importance sampling algorithm to social network analysis. We adopt the importance sampling algorithm to a continuous-time social network dynamic model. The algorithm guarantees that the samples generated from continuous-time social network dynamic models are consistent with the observations, which greatly improves the learning accuracy.

- We design the hidden social network dynamics model. The model allow us to use indirect observation data, such as communication events among people, to learn the dynamics of the social network dynamics, which is unobserved (hidden) all the time.

- We develop a Metropolis-Hastings sampling algorithm for the hidden social network dynamics model. The algorithms can handle both complete and partially observed data.

- We apply our sampling algorithms and hidden social network dynamics model to several real world applications. We demonstrate that social network dynamics can be learned from many indirect observations, which greatly extends the data types that can be used in analyzing social network dynamics.

Chapter 2

Continuous Time Markov Processes

In this chapter, we provide the background material about continuous-time Markov processes (see Norris [1997] for a more complete treatment). We first give the definition for a continuous-time, finite-state, homogeneous Markov process, and then discuss how to reason about continuous-time Markov processes.

2.1 Representation

Let X be a continuous-time, finite-state, homogeneous Markov process. X has n states $\{x_1, x_2, \ldots, x_n\}$. $X(t)$ is the (finite) state of the system at time t. The collection of random variables $\{X(t)|t \in \mathbb{R}^+\}$ composes the process. X satisfies the Markov assumption. That is,

$$P(X(t+s) = x_j|X(s) = x_i, X(r) = x_r) = P(x(t+s) = x_j|x(s) = x_i)$$

$\forall s, t \geq 0,\ 0 \leq r < s$ and $i, j \in \{1, \ldots, n\}$.

The initial distribution $P_X^0 = P(x(0))$ is a multinomial distribution over n states of X. The transient behavior of X is described by the initial distribution P_X^0 and the transition model which is often represented by the intensity matrix

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

where $q_{x_ix_j}$ is the intensity with which X transitions from x_i to x_j and $q_{x_i} = \sum_{j \neq i} q_{x_ix_j}$.

The diagonal elements q_i and the off-diagonal elements q_{ij} define the instantaneous transition probabilities of X. More precisely,

$$\lim_{\Delta t \to 0} \frac{P(X(t + \Delta t) = x_j | X(t) = x_i)}{\Delta t} = q_{ij}$$
$$\lim_{\Delta t \to 0} \frac{1 - P(X(t + \Delta t) = x_i | X(t) = x_i)}{\Delta t} = q_i \,.$$

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

$f(q_{x_i}, t)$ and the corresponding distribution function $F(q_{x_i}, t)$ for X staying in state x_i are

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

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

The expected time of transitioning is $1/q_{x_i}$. Upon transitioning, the probability that X transitions from state x_i to x_j is $\theta_{x_i x_j} = q_{x_i x_j}/q_{x_i}$.

Example 2.1.1 *Assume we want to model the behavior of a person's exercise intensity $E(t)$ which has three values ($Val(E(t)) = \{e_o = light, e_1 = medium, e_2 = heavy\}$). We could represent the dynamics of $E(t)$ using the intensity matrix*

$$\mathbf{Q}_E = \begin{bmatrix} -1.8 & 1.6 & 0.2 \\ 0.5 & -1 & 0.5 \\ 0.5 & 1.5 & -2 \end{bmatrix}.$$

If we set the time unit to one month, this means that we expect the person changes his exercise intensity in $1/2 = 0.5$ months if his current intensity is heavy. When the intensity is changing, with probability $0.5/2 = 0.25$ the new value will be light and with probability $1.5/2 = 0.75$ the new value will be medium.

Often times, the dynamics system we are trying to model contain more than one variable. To model a multi-variable system, we first combine all variables into a single joint variable by enumerating all possible states of the variables. If the system has N variables X_i ($i =$

$1, \ldots, N$), and each variable contains S_i states, the total number of states of the joint process is $n = \prod_{i=1}^{N} S_i$ and the size of the intensity matrix for the joint process is n by n. As the number of variable increases, the size of the intensity matrix grows exponentially.

Example 2.1.2 *Assume that in the previous example, we also need to consider the effect of the weather $W(t)$ which has two values ($Val(W(t)) = \{w_0 = rainy, w_1 = sunny\}$). To model the dynamics of this system, we first list all the possible values of the joint variable: (w_0, e_0), (w_0, e_1), (w_0, e_2), (w_1, e_0), (w_1, e_1), (w_1, e_2). We then write the transition intensity of each pair of values into the joint intensity matrix.*

$$
\begin{array}{c}
\\
w_0e_0 \\
w_0e_1 \\
w_0e_2 \\
w_1e_0 \\
w_1e_1 \\
w_1e_2
\end{array}
\begin{bmatrix}
w_0e_0 & w_0e_1 & w_0e_2 & w_1e_0 & w_1e_1 & w_1e_2 \\
-3.5 & 0.7 & 0.3 & 2.5 & 0 & 0 \\
4.4 & -7.5 & 0.6 & 0 & 2.5 & 0 \\
8.9 & 1.1 & -12.5 & 0 & 0 & 2.5 \\
0.75 & 0 & 0 & -2.55 & 1.6 & 0.2 \\
0 & 0.75 & 0 & 0.55 & -1.75 & 0.45 \\
0 & 0 & 0.75 & 0.5 & 1.5 & -2.75
\end{bmatrix}
$$

Notice that elements representing $W(t)$ and $E(t)$ changing simultaneously are all zeros. This means that we assume any two variables cannot change their state at exactly the same time. Since we are modeling a continuous-time dynamics, this is a natural assumption.

2.2 Likelihood and Sufficient Statistics

Samples from a Markov process are often called trajectories, each of which consists of the starting value of the process and a sequence of events. Each event is a transition that denotes the new value x_i that variable X changes to at time t. The probability or likelihood of the data thus can be calculated using sufficient statistics of the trajectories.

Suppose we are given a complete trajectory σ generated from the Markov process $X(t)$. To calculate the likelihood, we can decompose the trajectory into n segments according to the transition time. We represent each segment as $d_i =< t_i, x_i, x_{i+1} >$ where x_i is the value of X before the transition, x_{i+1} is the value after the transition, and t_i is the time X spends in state x_i before transitioning. The likelihood of a segment d_i can be written as

$$L_X(d_i) = q_{x_i} \exp(-q_{x_i} t_i) \times \theta_{x_i x_{i+1}} .$$

The likelihood of the entire trajectory σ is the probability of the starting value x_0, times the multiplication of the likelihood of each segment $L_X(d_i)$:

$$\begin{aligned} L_X(\sigma) &= P_X^0(x_0) \prod_{i=1}^{n} L_X(d_i) \\ &= P_X^0(x_0) \prod_{i=1}^{n} q_{x_i} \exp(-q_{x_i} t_i) \times \theta_{x_{d_i} x_{d_{i+1}}} . \end{aligned}$$

We summarize the data using the sufficient statistics:

- $T[x]$, the total amount of time X spends in state x, and

- $M[x, x']$, the total number of times X transitions from state x to state x', where $x \neq x'$.

The likelihood of the trajectory σ can be written as

$$L_X(\sigma) = P_X^0(x_0) \prod_x \left(q_x^{M[x]} \exp(-q_x T[x]) \times \prod_{x \neq x'} \theta_{xx'}^{M[x,x']} \right) .$$

2.3 Query of Markov Process

Given the intensity matrix $\mathbf{Q}_X$ and the initial distribution P_x^0 of a continuous-time homogeneous Markov process X, there are a number of questions we can answer.

Commonly we wish to calculate the conditional probability $P(X(t) = x_t | X(s) = x_s)$, where x_t is the value of X at time t and x_s is the value of X at an earlier time s. If X has n states $\{x_1, \ldots, x_n\}$, all the conditional probabilities can be represented using an $n \times n$ matrix $P(X(t)|X(s))$, where the i^{th} row vector is the conditional distribution over the value of X at time t given $X(s) = x_i$. $P(X(t)|X(s))$ can be calculated as

$$P(X(t)|X(s)) = \exp(\mathbf{Q}_X(t-s)) .$$

Let $P_X(t)$ be a row vector representing the distribution over the values of X at time t. This distribution can be calculated as

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

where the initial distribution P_X^0 is represented as a row vector. The function $\exp()$ denotes the matrix exponential, which is defined as following for a matrix Q:

$$\exp(Q) = \sum_{k=0}^{\infty} \frac{Q^k}{k!}. \tag{2.1}$$

For a real matrix of size 2×2, the matrix exponential can be calculated exactly as

$$\exp(\begin{bmatrix} q_{11} & q_{12} \\ \\ q_{21} & q_{22} \end{bmatrix}) = \frac{1}{\Delta} \begin{bmatrix} m_{11} & m_{12} \\ \\ m_{21} & m_{22} \end{bmatrix},$$

where

$$\begin{aligned}
m_{11} &= e^{(q_{11}+q_{22})/2}(\Delta \cosh(\frac{1}{2}\Delta) + (q_{11} - q_{22})\sinh(\frac{1}{2}\Delta)) \\
m_{12} &= 2q_{12}e^{(q_{11}+q_{22})/2}\sinh(\frac{1}{2}\Delta) \\
m_{21} &= 2q_{21}e^{(q_{11}+q_{22})/2}\sinh(\frac{1}{2}\Delta) \\
m_{22} &= e^{(q_{11}+q_{12})/2}(\Delta \cosh(\frac{1}{2}\Delta) - (q_{11} - q_{12})\sinh(\frac{1}{2}\Delta))
\end{aligned}$$

and

$$\Delta = \sqrt{(q_{11} - q_{22})^2 + 4q_{12}q_{21}}$$

For general real matrices, the matrix exponential can only be calculated numerically.

2.4 Summary

Many real world systems evolve naturally in continuous time. Continuous-time Markov process allows us to realistically model and reason about these dynamic systems using continuous-time. However, the size of the state space of a continuous-time Markov process grows exponentially with the number of variables in the system, which makes this method infeasible for large systems. One solution for such state space explosion is to use models with structured state spaces in which the dynamics of each local variable can be represented more efficiently. A *Continuous time Bayesian network* (CTBN) is such a model that decomposes a Markov process using graphical representation. It allows us to model large dynamic systems more efficiently and reason about these systems in continuous time. We will discuss the CTBN model in more detail in the following chapter.

Chapter 3

Continuous Time Bayesian Network

As we have discussed in the previous chapter, a continuous-time Markov process suffers from state space explosion when handling large dynamic systems. A structured representation is needed to deal with systems with many variables. In this chapter, we desribe the *continuous time Bayesian networks* (CTBNs), which were first introduced by Nodelman et al. [2002]. CTBNs use a graphical representation to describe multi-variable continuous-time stochastic processes, which can model real world dynamic systems efficiently.

3.1 Structured Process Representation

In order to decompose a multi-variable dynamic system, we introduce a *conditional intensity matrix (CIM)* to describe the dynamics of local variables in a system. Let $\boldsymbol{X}$ be all the variables of the dynamic system we are trying to model. Let $X \in \boldsymbol{X}$ be one variable in the system and $\mathbf{U} \subset \boldsymbol{X}$ be a set of other variables. The conditional intensity matrix $\mathbf{Q}_{X|\mathbf{U}}$ for

variable X is defined as a set of intensity matrices $\mathbf{Q}_{X|\mathbf{u}}$, one for each instantiation $\mathbf{u}$ of the variable set $\mathbf{U}$. The evolution of X depends instantaneously on the values of the variables in $\mathbf{U}$. Using a CIM, we can model each local variable as an inhomogeneous Markov process, whose intensities are a function of the current values of a set of other variables.

Example 3.1.1 *Let us consider the dynamic system described in Example 2.1.2. Instead of using a single Markov process to represent the whole system, we can model the dynamics of each local variable separately by utilizing the dependecies among the variables. For example, it is natural to stipulate that the exercise intensity depends on the weather condition. Therefore, the dynamics of the exercise intensity can be described using two CIMs.*

$$\mathbf{Q}_{E|w_0} = \begin{bmatrix} -1.0 & 0.7 & 0.3 \\ 4.4 & -5.0 & 0.6 \\ 8.9 & 1.1 & -10.0 \end{bmatrix} \quad \mathbf{Q}_{E|w_1} = \begin{bmatrix} -1.8 & 1.6 & 0.2 \\ 0.55 & -1.0 & 0.45 \\ 0.5 & 1.5 & -2.0 \end{bmatrix}.$$

The behavior of variable $E(t)$ is now represented as an inhomogeneous Markov process, whose intensities depend on the current value of $W(t)$. When $W(t) = w_0$, the behavior of $E(t)$ is described using $\mathbf{Q}_{E|w_0}$. When $W(t) = w_1$, it is described using $\mathbf{Q}_{E|w_1}$.

3.2 The CTBN Model

Using CIMs, we can explore the dependencies among variables and model each local variable as an inhomogeneous Markov process. Thus, we can represent a multi-variable dynamic system using a structured model, in which relations among variables can be described using

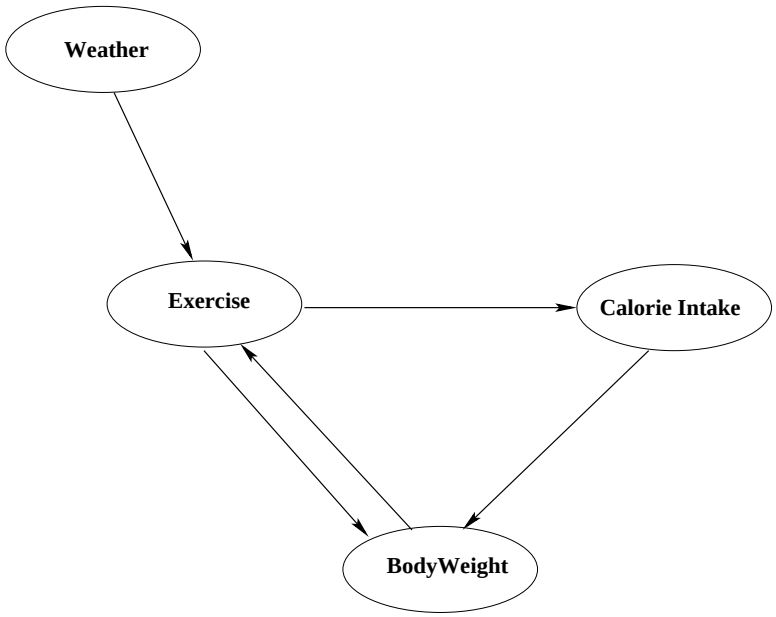


Figure 3.1: CTBN Example: Weight Control Effect

a directed graph. Each node in the graph represents a variable and each arc between two nodes represents the dependency between those two variables.

3.2.1 Model Definition

Definition 3.2.1 *[Nodelman et al., 2002] A* continuous time Bayesian network $\mathcal{N}$ *over* $\boldsymbol{X}$ *consists of two components: an* initial distribution $P^0_{\boldsymbol{X}}$*, specified as a Bayesian network* $\mathcal{B}$ *over* $\boldsymbol{X}$*, and a* continuous transition model*, specified using a directed (possibly cyclic) graph* $\mathcal{G}$ *whose nodes are* $X \in \boldsymbol{X}$*. Let* $\mathbf{U}_X$ *denote the parents of* X *in* $\mathcal{G}$*. Each variable* $X \in \boldsymbol{X}$ *is associated with a conditional intensity matrix,* $\mathbf{Q}_{X|\mathbf{U}_X}$*.*

Example 3.2.2 *Assume we want to model the behavior of a person controlling his body weight. When the person is overweight, he may exercise more to lose the excess weight.*

Increasing exercise intensity tends to increase his appetite, which will increase his daily calorie intake. Both exercise intensity and calorie intake contribute to his body weight. Furthermore, his exercise intensity also depends on the weather. Such a dynamic system contains four variables: body weight, exercise, calorie intake, and weather. Each variable changes in continuous time and its changing rate depends on the current value of some other variables.

We can use a CTBN to represent such behavior. The dependencies of these four variables are depicted using a graphical structure, as shown in Figure 3.1. The quantitative transient dynamics for each variable are represented using CIMs. Let's assume all the four variables are binary. Let $B(t)$ be the person's body weight ($Val(B(t)) = \{b_0 = normal, b_1 = overweight\}$), $E(t)$ be the exercise intensity ($Val(E(t) = \{e_0 = light, e_1 = heavy\}$), $C(t)$ be his daily calorie intake ($Val(C(t) = \{c_0 = low, c_1 = high\}$), and $W(t)$ be the weather ($Val(W(t) = \{w_0 = rainy, w_1 = sunny\}$). The conditional intensity matrices for the four variables can be specified as

$\mathbf{Q}_W$	$\mathbf{Q}_W = \begin{bmatrix} -0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix}$	
$\mathbf{Q}_{E\|W,B}$	$\mathbf{Q}_{E\|w_0,b_0} = \begin{bmatrix} -0.1 & 0.1 \\ 2 & -2 \end{bmatrix}$	$\mathbf{Q}_{E\|w_1,b_0} = \begin{bmatrix} -0.3 & 0.3 \\ 1 & -1 \end{bmatrix}$
	$\mathbf{Q}_{E\|w_0,b_1} = \begin{bmatrix} -0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix}$	$\mathbf{Q}_{E\|w_1,b_1} = \begin{bmatrix} -1 & 1 \\ 0.1 & -0.1 \end{bmatrix}$

$\mathbf{Q}_{C\|E}$	$\mathbf{Q}_{C\|e_0}$	=	$\begin{bmatrix} -0.2 & 0.2 \\ 1 & -1 \end{bmatrix}$	$\mathbf{Q}_{C\|e_1}$	=	$\begin{bmatrix} -1 & 1 \\ 0.2 & -0.2 \end{bmatrix}$
$\mathbf{Q}_{B\|E,C}$	$\mathbf{Q}_{B\|e_0,c_0}$	=	$\begin{bmatrix} -0.2 & 0.2 \\ 0.8 & -0.8 \end{bmatrix}$	$\mathbf{Q}_{B\|e_1,c_0}$	=	$\begin{bmatrix} -0.1 & 0.1 \\ 1 & -1 \end{bmatrix}$
	$\mathbf{Q}_{B\|e_0,c_1}$	=	$\begin{bmatrix} -1 & 1 \\ 0.1 & -0.1 \end{bmatrix}$	$\mathbf{Q}_{B\|e_1,c_1}$	=	$\begin{bmatrix} -0.2 & 0.2 \\ 0.6 & -0.6 \end{bmatrix}$

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

3.2.2 Conditional Independencies

By using the graphical model, CTBNs not only provide a structured representation language for multi-variable Markov processes, but also describe the independencies (and dependencies) among variables. Since a CTBN represents continuous-time dynamic systems, the independencies specified by a CTBN are between distributions over entire trajectories of the

variables. Similar to a Bayesian Network, each variable X_i in a CTBN is separated from all the other variables by its Markov blanket. The Markov blanket of variable X_i is defined as the parents of X_i, the children of X_i, and the children's parents. In a CTBN, X_i is independent of all the other variables in the CTBN given the entire trajectory of the Markov blanket of a variable X_i.

Example 3.2.3 *In the CTBN model showed in Figure 3.1, variable $C(t)$'s Markov blanket is $B(t)$ and $E(t)$. Therefore, $C(t)$ and $W(t)$ are separated by $B(t)$ and $E(t)$. That is, given the entire trajectory of $B(t)$ and $E(t)$, $C(t)$ and $W(t)$ are independent.*

3.2.3 Joint Markov Process

Although each node in a CTBN model represents an inhomogeneous Markov process, the behavior of the entire CTBN model still represents a single homogeneous Markov process. The intensity matrix $\mathbf{Q}$ for this joint Markov process is generated by combining all the CIMs in the CTBN model together, which is called *amalgamation* [Nodelman et al., 2002]. A basic assumption of CTBNs is that two variables cannot transition at exactly the same time. Therefore, all the elements in the joint intensity matrix that reflect two variables changing simutaneously should be zeros. If each variable X_i in the CTBN $\mathcal{N}$ has n_i states, the number of states of the joint Markov process is $n = \prod n_i$ and $\mathbf{Q}$ is an $n \times n$ matrix. We generate the joint intensity matrix $\mathbf{Q}$ by three steps.

- Let i and j be any pair of states in the joint Markov process where there is only one variable whose value is different between them. Let X be that different variable and $\mathbf{U}$

be the set of parents of X. Let u be the instantiation of $\mathbf{U}$ in state i and j. The value of the off-diagonal element q_{ij} in $\mathbf{Q}$ is set to be the corresponding intensity in the CIM $\mathbf{Q}_{X|\mathbf{u}}$.

- All the other off-diagonal elements are zero since two variables cannot transition at exactly the same time in a CTBN.

- The diagonal elements are computed to make each row sum to zero.

Example 3.2.4 *For CTBN model in Figure 3.1, we can amalgamate all the CIMs and form a homogeneous continuous-time Markov process. The joint Markov process has 16 states:* $x_1 = (w_0, e_0, c_0, b_0)$, $x_2 = (w_1, e_0, c_0, b_0)$, $x_3 = (w_0, e_1, c_0, b_0)$, ..., $x_{16} = (w_1, e_1, c_1, b_1)$. *Therefore, the intensity matrix for the joint Markov process is of size* 16×16. *The value of each off-diagonal element for which only one variable value is different between any two states is set to be the corresponding intensity in the CIM of that variable. For example, the value of the* $q_{x_1x_3}$ *represents changing from state* (w_0, e_0, c_0, b_0) *to state* (w_0, e_1, c_0, b_0). *We set it to be 0.1 according to the conditional intensity matrix* $\mathbf{Q}_{E|w_0,b_0}$. *All the other off-diagonal elements are set to be zeroes. The resulting matrix is*

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

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

3.3 Sufficient Statistics and Likelihood

The probability density over trajectories σ of a set of variables $\boldsymbol{X}$ described by a CTBN belongs to the exponential family. Therefore, similar to likelihood of a single Markov process in Section 2.2, the distribution of a CTBN can be described in terms of the sufficient statistics of σ [Nodelman et al., 2003]. In a CTBN, each variable $X \in \boldsymbol{X}$ is conditioned on its parent set $\mathbf{U}$. We can define the sufficient statistics of a CTBN as

- $T[x|\mathbf{u}]$, the amount of time $X = x$ while $\mathbf{U}_X = \mathbf{u}$, and

- $M[x, x'|\mathbf{u}]$, the number of transitions from x to x' while $\mathbf{U}_X = \mathbf{u}$.

If we let $M[x|\mathbf{u}] = \sum_{x'} M[x, x'|\mathbf{u}]$, the likelihood of each local variable X is

$$L_X(T[X|\mathbf{U}_X], M[X|\mathbf{U}_X]) = \prod_{\mathbf{u}} \prod_{x} \left(q_{x|\mathbf{u}}^{M[x|\mathbf{u}]} \exp(-q_{x|\mathbf{u}} T[x|\mathbf{u}]) \prod_{x' \neq x} \theta_{xx'|\mathbf{u}}^{M[x,x'|\mathbf{u}]} \right) \quad (3.1)$$

The probability density of trajectory σ is

$$P_{\mathcal{N}}(\sigma) = P^0_{\boldsymbol{X}}(\sigma_0) \prod_{X \in \boldsymbol{X}} L_X(T[X|\mathbf{U}_X], M[X|\mathbf{U}_X]) \quad (3.2)$$

where $P^0_{\boldsymbol{X}}(\sigma)$ is the probability of the starting values of the variables in trajectory σ. Since the initial distribution of a CTBN is specified using a Bayesian Network $\mathcal{B}$ of $\boldsymbol{X}$, $P^0_{\boldsymbol{X}}(\sigma)$ is the product of the conditional probability of each variable in $\mathcal{B}$.

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

3.4 Inference in CTBNs

Given a CTBN model, we would like to use it to answer queries conditioned on observations. Or in many situations, we would like to estimate the parameters of a CTBN model based on some partially observed evidence. In both cases, we need to perform inference in CTBNs.

3.4.1 Evidence and Queries

Evidence for a CTBN is usually a partial trajectory, in which some values or transitions are missing for some variables during some time intervals. There are two common types of observations: point evidence and continuous evidence. Point evidence represents the observation of the value of some variables at a particular time instant. Continuous evidence provides the behavior of some variables throughout an interval $[t_1, t_2)$. For instance, $x = 1$ during the interval $[2, 3.5)$, or $x = 1$ from $t = 2$ to $t = 3$ and then x transitions to $x = 0$ at $t = 3$ and stays in that state until $t = 5$. We define $x[t_1 : t_2)$ to be the behavior of variable X on the interval $[t_1, t_2)$, $x[t_1 : t_2]$ to be the behavior of X on the interval $[t_1, t_2]$, and $x(t_1 : t_2]$ to be the behavior of X on the interval $(t_1, t_2]$.

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

3.4.2 Exact Inference in CTBNs

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

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

To perform exact inference, we first generate the intensity matrix $\mathbf{Q}$ for the joint homogeneous Markov process using the amalgamation method described in Section 3.2.3. We then incorporate the evidence into $\mathbf{Q}$ as following.

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

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

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

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

where $\sigma_{[t_i,t_j]}$ and $\sigma_{[t_i,t_j)}$ represent the trajectory during interval $[t_i, t_j]$ and $[t_i, t_j)$ respectively. $\boldsymbol{\alpha}_t$ and $\boldsymbol{\beta}_t$ each have one element for each state of the system.

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

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

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

$$P(X_t = k, \sigma_{[0,T)}) = \boldsymbol{\alpha}_{t_i} \exp(\mathbf{Q}_i(t - t_i))\boldsymbol{\Delta}_{k,k} \exp(\mathbf{Q}_i(t_{i+1} - t))\boldsymbol{\beta}_{t_{i+1}} \tag{3.3}$$

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

Sufficient statistics of data are often used to calculate probabilities regarding the data and estimate the parameters of the model. When the given data is only partially observed, we can use inference to calculate the *expected sufficient statistics*. Given a partially observed trajectory σ, we can divide the trajectory into intervals as above and calculate the expected sufficient statistics during each interval $[t_i, t_{i+1})$ as following.

The expected total amount of time that the joint Markov process spends in state j during interval $[t_i, t_{i+1})$ given the evidence is:

$$\frac{1}{C} \int_{t_i}^{t_{i+1}} \boldsymbol{\alpha}_{t_i} \exp(\mathbf{Q}_i(t - t_i)) \mathbf{\Delta}_{j,j} \exp(\mathbf{Q}_i(t_{i+1} - t)) \boldsymbol{\beta}_{t_{i+1}} dt \,, \tag{3.4}$$

where C is the normalization constant to guarantee that the summation of the total time the process spends on each state during interval $[t_i, t_{i+1})$ is $t_{i+1} - t_i$.

The expected number of times that the joint Markov process transitions from state j to state k and was not observed doing so is:

$$\frac{q_{jk}}{C} \int_{t_i}^{t_{i+1}} \boldsymbol{\alpha}_{t_i} \exp(\mathbf{Q}_i(t - t_i)) \mathbf{\Delta}_{j,k} \exp(\mathbf{Q}_i(t_{i+1} - t)) \boldsymbol{\beta}_{t_{i+1}} dt \tag{3.5}$$

For details of computing the expected sufficient statistics in a CTBN, see Nodelman et al. [2005b]

3.4.3 Approximate Inference in CTBN

As discussed in the previous section, exact inference in a CTBN requires generating a single joint intensity matrix over the entire state space and calculating the exponential of the matrix (Equation 2.1). As the number of variables increases, the size of the entire state space grows exponentially. Exact inference is intractable for large networks. Therefore, approximate inference method is used. Several approximate inference algorithms have been developed. Generally, they can be categorized into three types: *expectation propagation* [Minka, 2001] approaches, *mean field variational* approaches, and sampling approaches.

Since a CTBN model uses a graphical representation, inference methods based on message passing can be applied. Nodelman et al. [2005a] introduced an approximate inference method based on *expectation propagation*. The algorithm partitions the evidence trajectory into segments according to the transitions of the evidence. On each segment, marginal distributions of local variables can be calculated using the forward-backward algorithm described in Section 3.4.2. Messages (marginal distributions and *CIMs*) of each segment are passed among the local variables. The accuracy of the *expectation propagation* algorithm can be increased by refining the length of the segments. Saria et al. [2007] extended it to full belief propagation and provided a method to adapt the approximation quality.

An inference algorithm based on mean field was introduced by Cohn et al. [2009]. Given a CTBN with n variables and observations, the algorithm approximates the posterior of the process using n independent inhomogeneous Markov processes. Each inhomogeneous Markov process is represented by a family of functions. Then inference in the given CTBN

can be posed as an optimization problem, which is to find the function of each inhomogeneous Markov process to maximize the free energy. The optimization can be performed by numerically solving a set of ordinary differential equations for each inhomogeneous Markov process iteratively.

Another way to perform approximate inference in CTBNs is to use sampling. Sampling has several advantages. Usually, it is easy to implement and does not require complex numeric computations. It is an anytime algorithm: we can stop at any time and use the samples we have obtained to compute the answer.

Ng et al. [2005] developed a continuous-time particle filtering algorithm. However, it only handles point evidence on binary and ternary discrete variables using rejection sampling, and focuses primarily on the incorporation of evidence from the continuous-state part of the system.

El-Hay et al. [2008] provided another sampling algorithm for CTBNs using Gibbs sampling. The algorithm starts from an arbitrary trajectory that is consistent with the evidence. Then, in each iteration, it randomly picks one variable X and samples an entire trajectory for that variable by fixing the trajectory of all the other variables. Since only X is not fixed, the conditioned cumulative distribution $F(t)$ that X stays in one state less than t and the state transition probabilities can be calculated exactly using standard forward and backward propagation within the Markov blanket of X. The Gibbs sampling algorithm can handle any type of evidence and it provides an approach to sample from the exact posterior distribution given the evidence. However, the posterior distribution $F(t)$ can be an arbitrarily complex

function. To sample exactly from it, binary search has to be applied and $F(t)$ is repeatedly evaluated, which may affect the efficiency of the algorithm.

A different sampling-based approach using importance sampling (our work) was first presented in Fan and Shelton [2008]. The algorithm generates weighted samples to approximate the expectation of a function of a trajectory. In Chapter 4, we will discuss the detailed algorithm.

3.5 CTBN Parameter Estimation

If we know the graphical structure of a CTBN, the parameters (the conditional intensity matrices) of the model can be estimated using a set of trajectories $D = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$.

When the dataset D is complete, where each trajectory σ_i is a complete set of state transitions and the times at which they occurred, the parameters can be learned by maximizing the log-likelihood of the dataset [Nodelman et al., 2003]. According to Equation 3.1 and 3.2, the log-likelihood can be written as the sum of the log-likelihood for each local variable. By maximizing the log-likelihoods in Equation 3.1, the parameters can be derived as

$$\hat{q}_{x|\mathbf{u}} = \frac{M[x|\mathbf{u}]}{T[x|\mathbf{u}]}; \quad \hat{\theta}_{xx'|\mathbf{u}} = \frac{M[x,x'|\mathbf{u}]}{M[x|\mathbf{u}]}. \tag{3.6}$$

The expectation maximization (EM) algorithm [Dempster et al., 1977] can be used to find the maximum likelihood parameters [Nodelman et al., 2005b] when the dataset is incomplete. The EM algorithm begins with an arbitrary initial parameter assignment, and alter-

natively repeats the expectation step and maximization step until convergence. In the expectation step, for each trajectory $\sigma_i \in D$, the expected sufficient statistics $\bar{M}[x|\mathbf{u}]$, $\bar{M}[x, x'|\mathbf{u}]$ and $\bar{T}[x|\mathbf{u}]$ are computed using exact inference. In the maximization step, new parameters are computed according to Equation 3.6 as if the expected sufficient statistics were the true sufficient statistics.

The expected sufficient statistics can be calculated using any inference algorithms (exact inference or approximate inference). When a sampling algorithm is used, this is called the Monte Carlo EM [Wei and Tanner, 1990] algorithm.

3.6 Summary

A CTBN provides a structured representation to model large dynamic systems. It allows us to model a dynamic system in continuous time. Therefore, a lot of complex dynamic systems, for example social networks, can be naturally modeled using CTBNs. Inference for a CTBN is the task of estimating the distribution over trajectories given a partially observed evidence. It is used in both answering queries about distributions and calculating expected sufficient statistics to estimate parameters of a CTBN when the observation data is incomplete. Performing exact inference in a CTBN requires constructing a joint intensity matrix for the entire system and computing the exponential of the matrix, which is often intractable. Thus, approximate inference methods need to be applied in many situations. However, many existing approximate inference methods for CTBNs depend on complex numeric computa-

tions. In the following chapter, we will introduce our sampling algorithms for CTBNs based on importance sampling, which only require simple calculations and are very easy to implement.

Chapter 4

Importance Sampling for CTBNs

As we described in the previous chapter, exact inference in a CTBN can be performed by generating a single joint intensity matrix over the entire state space. As the number of states is exponential in the number of the nodes in the network, this approach is infeasible when the network size is large. Approximate inference algorithms, therefore, are often used. However, a lot of existing approximate algorithms for CTBNs depend on complex numeric computations. They are often very hard to implement.

In this chapter, we provide a sampling based algorithm for CTBNs using importance sampling. The algorithm generates a trajectory by sampling the transition time and transition state naturally from exponential and multinomial distributions, which is very easy to implement. We extend the importance sampling algorithm to particle filtering and smoothing algorithms. We also derive a Metropolis-Hastings algorithm for CTBN inference based on the importance sampling algorithm.

4.1 Forward Sampling

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

$$\hat{\mathbf{E}}_{\mathcal{N}}[g] = \frac{1}{M} \sum_{m=1}^{M} g(\sigma[m]) \ . \tag{4.1}$$

For example, let $g = \mathbf{1}\{x(5) = x_1\}$, where $\mathbf{1}\{expr\}$ is 1 if $expr$ is true and returns 0 otherwise. Then we could use the above formula to estimate $P_{\mathcal{N}}(x(5) = x_1)$. Or the function $g(\sigma)$ might count the total number of times that X transitions from x_1 to x_2 while its parent U has value u_1, allowing us to estimate the expected sufficient statistic $M[x_1, x_2|u_1]$. The algorithm for sampling a trajectory is shown in Figure 4.1. For each variable $X \in \boldsymbol{X}$, it maintains $x(t)$ — the state of X at time t — and *Time*(X) — the next potential transition time for X. The algorithm adds transitions one at a time, advancing t to the next earliest variable transition. When a variable X (or one of its parents) undergoes a transition, *Time*(X) is resampled from the new exponential waiting time distribution. We use $\mathbf{u}_X(t)$ to represent the instantiation to parents of X at time t.

If we want to obtain a conditional probability of a query given evidence, the situation is more complicated. We might try to use *rejection sampling*: forward sample to generate possible trajectories, and then simply reject the ones that are inconsistent with our evidence.

Procedure *CTBN-Sample*(t_{end})
1. $t \leftarrow 0$, $\sigma \leftarrow \emptyset$
2. For each variable $X \in \boldsymbol{X}$
   Choose state $x(0)$ according to $\boldsymbol{\theta}^{\mathcal{B}}_{X|\mathbf{pa}_{\mathcal{B}}(X)}$.

Loop:

3. For each variable X such that *Time*(X) is undefined:
   Choose Δt for next X transition from an exponential with parameter $q_{x(t)|\mathbf{u}_X(t)}$.
   Define *Time*(X) $\leftarrow t + \Delta t$
4. Let $X = \arg\min_{X \in \boldsymbol{X}}[$*Time*$(X)]$
5. If *Time*(X) $\geq t_{end}$ **return** σ
6. Update $t \leftarrow$ *Time*(X)
7. Choose $x(t)$, the next value of X, from the multinomial with parameters $\boldsymbol{\theta}_{x(t)|\mathbf{u}_X(t)}$.
   Add $\langle X \leftarrow x(t), t \rangle$ to σ.
   Undefine *Time*(X), and *Time*(Y) for all variables Y for which $X \in \mathbf{U}_Y$.

Figure 4.1: Forward sampling semantics for a CTBN

The remaining trajectories are sampled from the posterior distribution given the evidence, and can be used to estimate probabilities as in Equation 4.1. However, this approach is entirely impractical in our setting, as in any setting involving an observation of a continuous quantity — in our case, time. In particular, suppose we observe that X transitions from x_1 to x_2 at time t. The probability of sampling a trajectory in which that transition occurs at precisely that time is zero. Thus, if we have evidence about transitions, with probability 1, none of our sampled trajectories will be relevant.

4.2 Importance Sampling in CTBNs

A more practical approach to sampling in the presence of evidence is importance sampling [Hesterberg, 1995]. In this section, we introduce an importance sampling algorithm for CTBNs.

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

$$w(\sigma) = \frac{P_{\mathcal{N}}(\sigma, \mathbf{e})}{P'(\sigma)} \ . \tag{4.2}$$

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

$$\hat{\mathbf{E}}_{\mathcal{N}}[g \mid \mathbf{e}] = \frac{1}{W} \sum_{m=1}^{M} g(\sigma[m]) w(\sigma[m]) \tag{4.3}$$

where W is the sum of the weights.

This estimator is consistent if the support of P' is a superset of the support of $P_{\mathcal{N}}(\cdot|\mathbf{e})$. In general, $\hat{\mathbf{E}}_{\mathcal{N}}$ is biased and the bias deceases as $O(M^{-1})$. The variance of the estimator also decreases as $O(M^{-1})$.

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

4.2.1 Simple Evidence

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

According to Section 3.3, $x[t_1 : t_2)$ can be summarized by the sufficient statistics over X on the interval $[t_1, t_2)$. Let $\tilde{L}_X(x[t_1 : t_2))$ be a partial likelihood contribution function, computed by plugging the sufficient statistics of $x[t_1 : t_2)$ into Equation 3.1. The partial contribution function can be defined over a collection of intervals $\mathcal{I}$ as

$$\tilde{L}_X(\mathcal{I}) = \prod_{x[t_1 : t_2) \in \mathcal{I}} \tilde{L}_X(x[t_1 : t_2)).$$

Returning to our simple evidence above, let $\tau_1 < \tau_2 \ldots, \tau_{n-1} < \tau_n$ be all the transition times in $\sigma_{[0,T)}$, $\tau_0 = 0$ and $\tau_{n+1} = T$. The likelihood of drawing σ from the target distribution $P_{\mathcal{N}}$ is

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

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

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

To compute the proper weight $w(\sigma)$ we substitute in Equation 4.2, and get

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

Therefore, the weight $w(\sigma)$ is the likelihood contribution of all the variables in $\boldsymbol{V}$. This algorithm exactly corresponds to *likelihood weighting* in Bayesian networks [Shachter and Peot, 1989, Fung and Chang, 1989]. Intuitively, this makes sense because we can account for all the evidence by simply assigning the observed trajectories to the observed variables.

4.2.2 General Evidence

Now, consider a general evidence pattern $\mathbf{e}$, in which we have time instants where variables become observed or unobserved. How can we force our trajectory to be consistent with $\mathbf{e}$?

Suppose there is a set of variables which have evidence beginning at t_e. We cannot simply force a transition at time t_e to make the variables consistent with the evidence $\mathbf{e}$: if the set contains more than one variable, the sample would have multiple simultaneous transitions, an event whose likelihood is zero.

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

To calculate the weight $w(\sigma)$, we partition σ into two pieces. Let σ_e be the collection for all variables $X \in \boldsymbol{X}$ of intervals $x[t_1 : t_2)$ where the behavior of X is set by the evidence. Let σ_s be the complement of σ_e containing the collection of intervals of unobserved behavior for all variables. By applying Equation 4.2, we have

$$
\begin{aligned}
w(\sigma) &= \frac{P_{\mathcal{N}}(\sigma, \mathbf{e})}{P'(\sigma)} \\
&= \prod_{x[\tau_i : \tau_{i+1}) \in \sigma_s} \frac{\tilde{L}_X(x[\tau_i : \tau_{i+1}))}{\tilde{L}'_X(x[\tau_i : \tau_{i+1}))} \times \prod_{x[\tau_i : \tau_{i+1}) \in \sigma_e} \frac{\tilde{L}_X(x[\tau_i : \tau_{i+1}))}{\tilde{L}'_X(x[\tau_i : \tau_{i+1}))} \\
&= \prod_{x[\tau_i : \tau_{i+1}) \in \sigma_s} \frac{\tilde{L}_X(x[\tau_i : \tau_{i+1}))}{\tilde{L}'_X(x[\tau_i : \tau_{i+1}))} \times \prod_{x[\tau_i : \tau_{i+1}) \in \sigma_e} \tilde{L}_X(x[\tau_i : \tau_{i+1})) \qquad (4.4)
\end{aligned}
$$

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

- σ_{sn} be the collection for all variables $X \in \boldsymbol{X}$ of intervals $x[t_1 : t_2)$ where the transition time is sampled from an exponential distribution.

- σ_{st} be the collection for all variables $X \in \boldsymbol{X}$ of intervals $x[t_1 : t_2)$ where the transition time is sampled from a truncated exponential distribution and the variable is involved in the next transition.

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

Therefore, we can rewrite Equation 4.4 as

$$w(\sigma) = \prod_{x[\tau_i : \tau_{i+1}) \in \sigma_{sn}} \frac{\tilde{L}_X(x[\tau_i : \tau_{i+1}))}{\tilde{L}'_X(x[\tau_i : \tau_{i+1}))} \times \prod_{x[\tau_i : \tau_{i+1}) \in \sigma_{st}} \frac{\tilde{L}_X(x[\tau_i : \tau_{i+1}))}{\tilde{L}'_X(x[\tau_i : \tau_{i+1}))} \times \prod_{x[\tau_i : \tau_{i+1}) \in \sigma_{sf}} \frac{\tilde{L}_X(x[\tau_i : \tau_{i+1}))}{\tilde{L}'_X(x[\tau_i : \tau_{i+1}))} \times \prod_{x[\tau_i : \tau_{i+1}) \in \sigma_e} \tilde{L}_X(x[\tau_i : \tau_{i+1})) \quad (4.5)$$

Example 4.2.1 *Assume that we are given a CTBN with two binary variables X and Y. X has two states x_0 and x_1. Y has two states y_0 and y_1. We have the observation that X is x_1 in interval $[t_1, t_2)$ and $[t_3, T)$, as shown in Figure 4.2(a). To answer queries based on the evidence, we use the method above to sample trajectories. Figure 4.2(b) shows one of the sampled trajectories. To calculate the weight of the trajectory, we partition the trajectory into four categories (as shown in Figure 4.2(c) and Figure 4.2(d)), and apply Equation 4.5.*

According to Equation 4.5, each time we add a new transition to the trajectory, we advance time from t to $t + \Delta t$. For each variable x we must update the weight of the trajectory to reflect the likelihood ratio for $x[t : t + \Delta t]$ based on the distribution we use to sample the "next time" and the transition variable we select. Each such variable can be considered separately as their times are sampled independently.

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

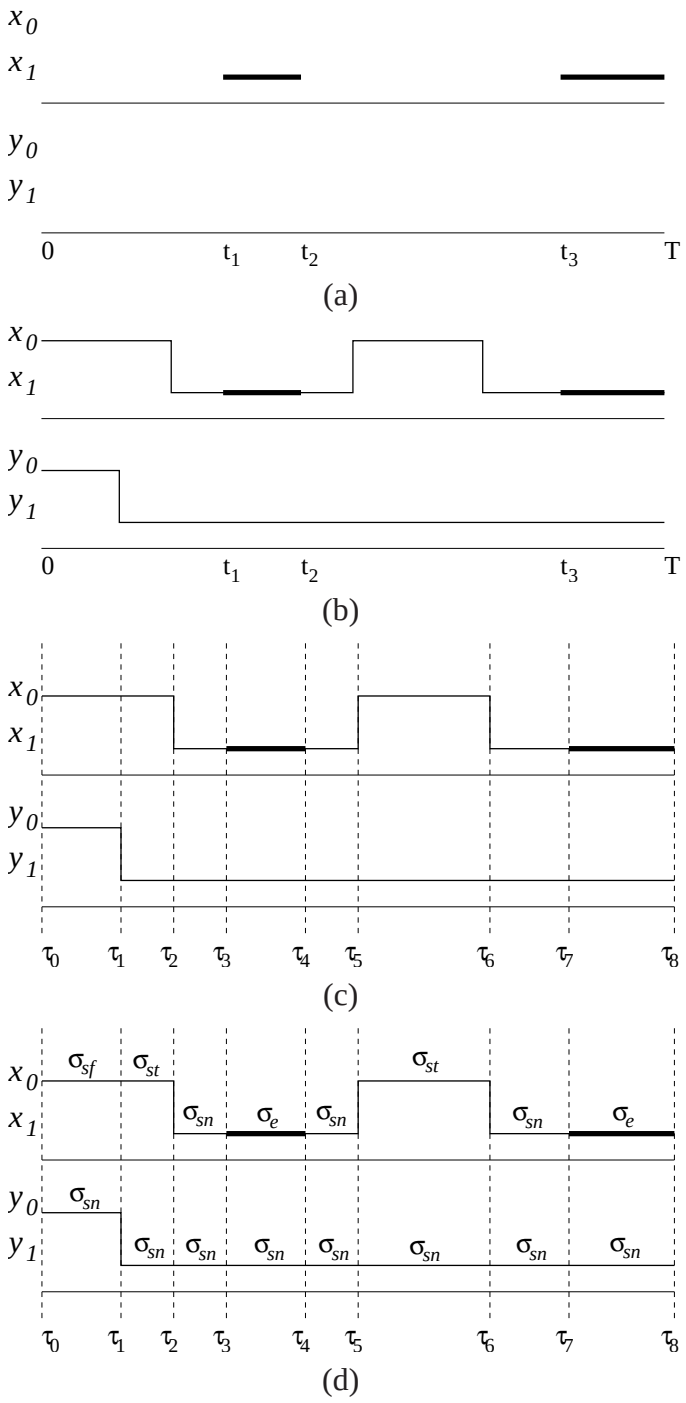


Figure 4.2: Example of calculating weight contribution. (a) Evidence of a CTBN. (b) A sampled trajectory agreeing with the evidence. (c). Partitioning of the trajectory according to the evidence and the transitions. σ_e equals $x[\tau_3 : \tau_4)$ and $x[\tau_7 : \tau_8)$ (d) Partitioning of the trajectory based on the different sampling situations.

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

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

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

The algorithm for CTBN importance sampling is shown in Figure 4.3. To more easily describe the evidence, we define a few helper functions:

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

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

Procedure *CTBN-Importance-Sample*$(t_{end}, \mathbf{e})$

1. $t \leftarrow 0$, $\sigma \leftarrow \emptyset$, $w \leftarrow 1$ *
2. For each variable $X \in \boldsymbol{X}$
   If $\mathbf{e}_X^{val}(0)$ defined,
      set $x(0) \leftarrow \mathbf{e}_X^{val}(0)$, *
      Set $w \leftarrow w \cdot \theta^{\mathcal{B}}_{x(0)|\mathbf{pa}_{\mathcal{B}}(0)}$ *
   Else choose state $x(0)$ according to $\boldsymbol{\theta}^{\mathcal{B}}_{X|\mathbf{pa}_{\mathcal{B}}(X)}$

Loop:

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

Procedure *Update-Weight*(Y, w, t_1, t_2)

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

Figure 4.3: Importance sampling for CTBNs. Changes from Figure 4.1 are noted with asterisks.

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

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

In Figure 4.3, the line numbers follow those given in the forward sampling algorithm with new or changed lines marked with an asterisk. *Time*(X) might be set to the end of an interval of evidence which is not a transition time but simply a time when we need to resample a next potential transition. This means that we will not update σ with a new transition every time through the loop. The algorithm differs from the forward sampling procedure as follows. Step 2 now accounts for evidence at the beginning of the trajectory (using standard likelihood weighting for Bayesian networks). In Step 3, we draw Δt from the truncated exponential if the current value disagrees with upcoming evidence. If the current evidence includes this variable, Δt is set to the duration of such evidence. Step 5 updates the weights using the procedure *Update-Weight*. Finally, Step 7 now deals with variables that are just leaving the evidence set.

4.2.3 Predictive Lookahead

The algorithm in Figure 4.3 draws the next state for a variable from the same distribution as the forward sampling algorithm. This may cause a variable to transition several times in a short interval before evidence as the variable "searches" to find a way to transition into the evidence. Thus, we may generate many unlikely samples, making the algorithm inefficient.

We can help mitigate this problem by trying to force the variable into a state that will lead to the evidence.

When sampling the next state for variable X at time t, instead of sampling from the multinomial according to $\boldsymbol{\theta}_{x(t)|\mathbf{u}_X(t)}$, we would like to sample from the distribution of the next state conditioned on the upcoming evidence. Suppose X is in state x_i at time t, and the next evidence for X is state x_k at t_e. Assuming the parents of X do not change before t_e and ignoring evidence over the children of X, the distribution of the state of X at t given only the evidence can be calculated using Equation 3.3:

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

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

4.3 Particle Filtering

The algorithm in Figure 4.3 allows us to generate a single trajectory and its weight, given the evidence. To apply this algorithm to the task of online inference in a dynamic system, we can generate multiple trajectories in parallel, advancing time forward as evidence is obtained.

The resulting algorithm is an instance of sequential importance sampling, and therefore suffers from its characteristic flaw: As the trajectory length increases, the distribution of the importance weights gets increasingly skewed, with most importance weights converging to zero exponentially quickly. Thus, the number of "relevant" samples gets increasingly small, and the estimates provided by the set of samples quickly become meaningless. A family of methods, commonly known as sequential Monte Carlo or particle filtering [Doucet et al., 2001], have been proposed in the setting of discrete-time processes to address this flaw. At a high level, these methods re-apportion our samples to focus effort on the more relevant samples — those with higher weight.

The application of this idea to our setting introduces some subtleties because different samples are not generally synchronized. We could pick a time t and run the algorithm in Figure 4.3 with $t_{end} = t$ so that samples are synchronized at t. We would re-apportion the weights and continue each trajectory from its state at t, first setting $\mathit{Time}(X)$ to be undefined for all X. However, choosing the proper synchronization time t is a non-trivial problem which may depend on the evidence and the speed the system evolves.

Instead of synchronizing all the particles by the time, we can align particles by the number of transitions. If we let t_i be the i^{th} transition time, X_i be the value of X from t_{i-1} to t_i, $t_{1:n}$ be the sequence of transition time t_i, and $X_{1:n}$ be the sequence of values of X_i, $(i = 1, \ldots, n)$, the following recursion holds.

Procedure *CTBN-Particle-Filtering*$(\{X_0^i, w_0^i\}_{i=1...N}, t_{end}, \mathbf{e})$

1. $k \leftarrow 0, W_t \leftarrow 1, N_r \leftarrow N$
2. For $i \leftarrow 1$ to N: $Pa_0^i \leftarrow i$, $w^i \leftarrow 1/N$

Loop:

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

Figure 4.4: Particle Filtering for CTBNs

$$P(X[0:t_n)) = P(X_{1:n}, t_{1:n}, e_{[0:t_n)})$$

$$= P(X_{1:n-1}, t_{1:n-1}, e_{[0:t_{n-1})}) P(X_n|X_{n-1}) P(t_n, e_{[t_{n-1},t_n)}|t_{n-1}, X_n)$$

The weighted approximation of this probability is given by

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

where $X^i[0:t_n)$ is the i^{th} sample and $w(X^i[0:t_n))$ is the normalized weight of the i^{th}sample. According to Equation 4.5, the weight can be updated after every transition step. The weight update equation can be shown as

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

Thus, to sample multiple trajectories in parallel, we apply the CTBN importance sampling algorithm to each trajectory until a transition occurs. To avoid the degeneracy of the weights, we resample the particles when the estimated effective sample size $\widehat{N_{eff}} = \frac{1}{\sum_i (w_k^i)^2}$ is below a threshold N_{thr}. This procedure is similar to the regular particle filtering algorithm except that all particles are not synchronized by time but the number of transitions. To answer queries in the time interval $[0, T)$, we propagate the particles until all of their last transitions are greater than T.

Figure 4.4 shows the algorithm for generating N trajectories from 0 to T in a CTBN. It assumes that the initial values and the weights have already been sampled. The procedure *Sample-Segment* loops from line 3 to 7 in Figure 4.3 until a transition occurs, returns the transition time and variables value, and updates the corresponding weight for that segment. Note that we are approximating the distribution $P(X_{1:n}, t_{1:n}, e_{[0:t_n)})$ for all possible n. Therefore, we only propagate and re-apportion weights for particles that have not yet reached time T. Particles that have been sampled past T are left untouched.

4.4 Particle Smoothing

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

The smoothing algorithm for discrete-time systems generates trajectories using N weighted particles $\{x_t^i, w_t^i\}$ from the particle filtering algorithm. It starts with the particles at time T, moves backward one step each iteration, and samples a particle according to the product of its weight and the probability of it transitioning to the previously sampled particle. Specifically, in the first step, it samples $\widetilde{x}_T$ from particles x_T^i at time T with probability w_T^i. In the back-

ward smoothing steps it samples $\widetilde{x}_t$ according to $w^i_{t|t+1} = w^i_t f(\widetilde{x}_{t+1}|x^i_t)$, where $f(\widetilde{x}_{t+1}|x^i_t)$ is the probability that the particle transitions from state x^i_t to $\widetilde{x}_{t+1}$. The resulting trajectory set is an approximation of $P(x_{1:T}|y_{1:T})$ where $y_{1:T}$ is the observation.

This idea can be used in our setting with some modifications. Given the filtered particles $\{X^i_{m_i}, t^i_{m_i}, w^i_{m_i}\}$, we need to sample both variable values and transition time at each step when we move backward. There are two main differences from the algorithm in Godsill et al. [2004]. First, there are fewer than N particles that can be used at the beginning steps of the backward smoothing since the trajectories do not have exactly the same number of transitions. And second, not all particles at step n can be considered as candidates to move backward. A particle $\{X^i_n, t^i_n, w^i_n\}$ is a valid candidate as the predecessor for $\{\widetilde{X}_{n+1}, \widetilde{t}_{n+1}\}$ only if (1) $t^i_n < \widetilde{t}_{n+1}$, (2) the values of X^i_n and $\widetilde{X}_{n+1}$ differ in only one variable (thus a single transition is possible), and (3) $\mathbf{e}_{(t^i_n, \widetilde{t}_{n+1})}$ contains no transitions.

Figure 4.5 shows the smoothing algorithm which generates a trajectory from the filtering particles. We apply the algorithm N times to sample N trajectories. These equally weighted trajectories can be used to approximate the distribution $P(X_{[0,T)}|\mathbf{e})$. Generating one trajectory with this smoothing process requires considering all the particles at each step. The running time of sampling N trajectories using particle smoothing is on the order of N times of that of particle filtering.

Procedure *CTBN-Particle-Smoothing*$(\{X^i_{m_i}, t^i_{m_i}, w^i_{m_i}\}, t_{end}, \mathbf{e})$
$i = 1 \ldots N, m_i = 1 \ldots M_i$

1. $\sigma \leftarrow \emptyset$
2. Choose k with probability $w_i^{M_i}$
3. set $Y = X_{M_k^k}$, $s \leftarrow M_k$, $t \leftarrow t_s^k$

Loop:

4. $\sigma_{[t_{s-1},s)} \leftarrow Y$
5. If σ is complete
   return σ
6. For $j \leftarrow 1$ to N
   $w'_j \leftarrow$ *Check-Weight*$(Y, t, X^j_{s-1}, t^j_{s-1}, w^j_{s-1})$
7. Choose i with probability w'_i
8. $S \leftarrow S - 1$, $Y \leftarrow X^s_i$, $t \leftarrow t^j_s$

Procedure *Check-Weight*(X, t, X_s, t_s, w_s)

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

Figure 4.5: Particle Smoothing for CTBNs

4.5 Markov Chain Monte Carlo for CTBNs

As we discussed above, importance sampling has the problem that the distribution of the importance weights gets increasingly skewed as the length of the trajectory increases. An alternative method to solve this problem is *Markov chain Monte Carlo (MCMC)*. The MCMC method generates a sequence of samples by running a carefully constructed Markov chain for a long time. The Markov chain is constructed so that successive samples in the sequence are drawn from distributions that gets closer and closer to the target distribution π.

In particular, assume we want to generate samples from a target distribution π over variables $\boldsymbol{X}$ with state space $Val(\boldsymbol{X})$. We construct a Markov chain in which each state of the chain is an instantiation of $\boldsymbol{X}$, and thus the state space of the Markov chain is all the possible instantiations of $\boldsymbol{X}$. The transition model $\mathcal{T}(x \rightarrow x')$ is designed so that the stationary distribution of the Markov chain is the target distribution π. We simulate the Markov chain as follows. We start the Markov chain with a random generated state. Then at each step, we generate the next state by picking a variable $X_i \in \boldsymbol{X}$ randomly, forgetting the current value of X_i, fixing the values of the rest of the variables, and sampling the new value of X_i according to the transition probability $\mathcal{T}$. Therefore, in each step, we generate a sample of $\boldsymbol{X}$ from the Markov chain. If we run the Markov chain for a long time, the state distribution gets closer and closer to the stationary distribution, which is the target distribution π. Then we can consider that samples we generate from the Markov chain are generated from π.

In a CTBN, each sample is a trajectory over all the variables of the CTBN. To apply MCMC method, we need to construct a Markov chain in which each state represents a possible trajectory of the CTBN. Thus, a state transition in the Markov chain consists of removing the entire trajectory of one variable and replacing it with a newly sampled one. Specifically, suppose we want use MCMC algorithm to sample trajectories from a CTBN $\mathcal{N}$ with n variables $\boldsymbol{X} = (X_i, X_2, \ldots, X_n)$ given the evidence $\mathbf{e}$. We construct a Markov chain whose state space is all the possible trajectories of $\mathcal{N}$ that are consistent with the evidence $\mathbf{e}$. We start the Markov chain with an arbitrary trajectory that is consistent with the evidence. In each step, the sampler randomly picks one variable X_i, fixes the trajectories of the other variables $Y = \{X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n\}$ as evidence, and samples the entire trajectory of X_i according to the transition model $\mathcal{T}(\sigma \rightarrow \sigma')$, where σ is the current state of the MCMC sampler (trajectory of the CTBN) and σ' is the newly generated MCMC state.

Different choices of the transition model $\mathcal{T}$ of the Markov chain result in different MCMC algorithms. In this section, we introduce a Gibbs sampling algorithm for CTBNs due to El-Hay et al. [2008] and a Metropolis-Hastings algorithm based on our importance sampling algorithm for CTBNs.

4.5.1 Gibbs Sampling

El-Hay et al. [2008] recently provided a Markov Chain Monte Carlo (MCMC) procedure which used a Gibbs sampler to generate samples from the posterior distribution given the evidence. Gibbs sampling defines the transition probability $\mathcal{T}$ to be the posterior distribution

of the selected variable X_i given the rest of the trajectory and evidence $\mathbf{e}$. More precisely, let σ_{X_i} be the trajectory of X_i in σ, σ'_{X_i} be the trajectory of X_i in σ', and σ_Y be the trajectory of the rest of the variables. The transition model is defined as

$$\mathcal{T}(\sigma \to \sigma') = \mathcal{T}((\sigma_{X_i}, \sigma_Y) \to (\sigma'_{X_i}, \sigma_Y)) = P(\sigma'_{X_i}|\sigma_Y, \mathbf{e}) \tag{4.6}$$

To generate the entire trajectory of X_i from the posterior distribution $P(\sigma'_{X_i}|\sigma_Y, \mathbf{e})$, the states and transitions of X_i need to be sampled in those intervals that X_i is not observed according to the evidence. The trajectory in each unobserved interval of X_i can be generated by alternatively sampling a transition time Δt and a new state x from the posterior distribution given $\mathbf{e}$ and the trajectories of the other variables Y.

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

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

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

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

where

$$\tilde{\alpha}(t) = Pr(X_i(0:t] = x_0, Y[0:T] | X_i(0) = x_0, Y_0)$$

$$\tilde{\beta}_x(t) = Pr(X_i(T) = x_T, Y(t:T] | X_i(t) = x, Y(t))$$

$\tilde{\alpha}(t)$ and $\tilde{\beta}_x(t)$ can be calculated using a slightly modified version of the standard forward-backward algorithm described in Section 3.4.2. Using the fact that X_i is independent of all the other components given the entire trajectory of its Markov blanket, the computation of $\tilde{\alpha}(t)$ and $\tilde{\beta}(t)$ can be limited to X_i and its Markov blanket (the parents of X_i, the children of X_i, and the children's parents).

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

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

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

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

4.5.2 Metropolis-Hastings Algorithm

Alternatively, we can use the Metropolis-Hastings algorithm which allows us define the transition model in a more general way. Unlike the Gibbs sampling algorithm, the Metropolis-Hastings algorithm does not require the transition distribution of the Markov chain to be the posterior distribution $P(\sigma'_{X_i}|\sigma_Y, \mathbf{e})$. Instead, it uses a proposal distribution $\mathcal{T}^Q$ as the transition model. In each step, we do not simply accept the new state σ' generated by the proposal distribution. We either reject it and stay at the old state σ, or accept it with acceptance probability $\mathcal{A}(\sigma \to \sigma')$. The acceptance probability is defined as

$$\mathcal{A}(\sigma \to \sigma') = \min\left[1, \frac{\pi(\sigma')\mathcal{T}^Q(\sigma' \to \sigma)}{\pi(\sigma)\mathcal{T}^Q(\sigma \to \sigma')}\right]. \tag{4.7}$$

Now, in each step when we pick a variable X_i to sample, instead of sampling the trajectory of X_i from $P(\sigma'_{X_i}|\sigma_Y, \mathbf{e})$, we can use our importance sampling algorithm considering $\mathbf{e}$ and the trajectory of all the other variables as evidence. That is, we choose the transition model as the proposal distribution defined by the importance sampling algorithm. Therefore, we have

$$\mathcal{T}^Q(\sigma' \to \sigma) = P'(\sigma'_{X_i}|\sigma_{\mathbf{U}_{X_i}}, \mathbf{e}_{X_i}), \tag{4.8}$$

where σ'_{X_i} is the new trajectory we sampled for X_i using the importance sampling algorithm, $\sigma_{\mathbf{U}_{X_i}}$ is the trajectory of the parents of X_i and $\mathbf{e}_{X_i}$ is the evidence of X_i.

Using Equation 4.8, we can calculate the acceptance ratio $\frac{\pi(\sigma')\mathcal{T}^Q(\sigma' \to \sigma)}{\pi(\sigma)\mathcal{T}^Q(\sigma' \to \sigma)}$ as the following:

$$\begin{aligned}
&\frac{\pi(\sigma')\mathcal{T}^Q(\sigma \to \sigma')}{\pi(\sigma)\mathcal{T}^Q(\sigma' \to \sigma)} \\
=&\frac{P_{\mathcal{N}}(\sigma')}{P_{\mathcal{N}}(\sigma)} \frac{P'(\sigma_{X_i}|\sigma_{\mathbf{U}_{X_i}}, \mathbf{e}_{X_i})}{P'(\sigma'_{X_i}|\sigma_{\mathbf{U}_{X_i}}, \mathbf{e}_{X_i})} \\
=&\frac{P_{\mathcal{N}}(\sigma')}{P_{\mathcal{N}}(\sigma)} \frac{P_{\mathcal{N}}(\sigma'_{X_i}, \mathbf{e}_{X_i}|\sigma_{\mathbf{U}_{X_i}})}{P'(\sigma'_{X_i}|\sigma_{\mathbf{U}_{X_i}}, \mathbf{e}_{X_i})} \frac{P'(\sigma_{X_i}|\sigma_{\mathbf{U}_{X_i}}, \mathbf{e}_{X_i})}{P_{\mathcal{N}}(\sigma_{X_i}, \mathbf{e}_{X_i}|\sigma_{\mathbf{U}_{X_i}})} \frac{P_{\mathcal{N}}(\sigma_{X_i}, \mathbf{e}_{X_i}|\sigma_{\mathbf{U}_{X_i}})}{P_{\mathcal{N}}(\sigma'_{X_i}, \mathbf{e}_{X_i}|\sigma_{\mathbf{U}_{X_i}})}
\end{aligned} \tag{4.9}$$

According to Equation 4.2, $\frac{P_{\mathcal{N}}(\sigma'_{X_i}, \mathbf{e}_{X_i}|\sigma_{\mathbf{U}_{X_i}})}{P'(\sigma'_{X_i}|\sigma_{\mathbf{U}_{X_i}}, \mathbf{e}_{X_i})}$ is the weight contribution $w(\sigma'_{X_i})$ of variable X_i when we use importance sampling algorithm to sample σ_{X_i} considering $\mathbf{e}$ and the trajectory of all the other variables of σ as evidence. $\frac{P_{\mathcal{N}}(\sigma_{X_i}, \mathbf{e}_{X_i}|\sigma_{\mathbf{U}_{X_i}})}{P'(\sigma_{X_i}|\sigma_{\mathbf{U}_{X_i}}, \mathbf{e}_{X_i})}$ is the weight contribution of X_i as if σ_{X_i} were sampled using importance sampling algorithm. We denote it as $w(\sigma_{X_i})$.

Then Equation 4.9 can be written as

$$\begin{aligned}\frac{\pi(\sigma')\mathcal{T}^{Q}(\sigma' \to \sigma)}{\pi(\sigma)\mathcal{T}^{Q}(\sigma' \to \sigma)} =& \frac{L_{\mathcal{N}}(\sigma')}{L_{\mathcal{N}}(\sigma)} \cdot \frac{w(\sigma'_{X_i})}{w(\sigma_{X_i})} \cdot \frac{L_{X_i}(\sigma)}{L_{X_i}(\sigma')} \\ =& \frac{L_{Y}(\sigma')}{L_{Y}(\sigma)} \cdot \frac{w(\sigma'_{X_i})}{w(\sigma_{X_i})},\end{aligned} \tag{4.10}$$

where $L_{\mathcal{N}}(\cdot)$ is the likelihood of the whole CTBN, $L_{X_i}(\cdot)$ is the likelihood contribution of X_i and $L_Y(\cdot)$ is the likelihood contribution of all variables except X_i.

Replacing the acceptance ratio in Equation 4.7 with Equation 4.10, the acceptance probability of the Metropolis-Hastings algorithm for CTBNs is

$$\mathcal{A}(\sigma \to \sigma') = \min\left[1, \frac{L_Y(\sigma')}{L_Y(\sigma)} \frac{w(\sigma'_{X_i})}{w(\sigma_{X_i})}\right]. \tag{4.11}$$

$L_Y(\cdot)$ is the product of the likelihood contribution $L_X(\cdot)$ for each $X \in Y$, which can be calculated using Equation 3.1. The weight contribution $w(\sigma_{X_i})$ and $w(\sigma'_{X_i})$ are calculated according to Equation 4.5. Our Metropolis-Hastings sampling algorithm for CTBNs can be described as follows. We start an arbitrary trajectory that is consistent with the evidence $\mathbf{e}$. Each step, we randomly pick a variable X_i. We remove the trajectory of X_i from the current trajectory σ and generate a new trajectory σ' by using our importance sampling algorithm to sample trajectory for X_i, fixing all the other variables as evidence. We calculate the acceptance ratio according to Equation 4.10. If the ratio is larger than 1, we accept σ'. Otherwise, accept σ' with probability equal to the acceptance ratio, or reject σ' and keep σ.

4.6 Experimental Results

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

All the algorithms we used in the experiments were implemented in the same code base to make fair comparisons. We tried our best to optimize all the code. The implementations are general so that they can be applied to any CTBN model. Our implementation of expectation propagation is adapted from that of Saria et al. [2007] who were kind enough to share their code.

4.6.1 Networks

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

Drug Effect Network: The drug effect network is a toy model of the effect of a pain-relief medicine. It has 8 (5 binary and 3 ternary) variables. The structure of the network is

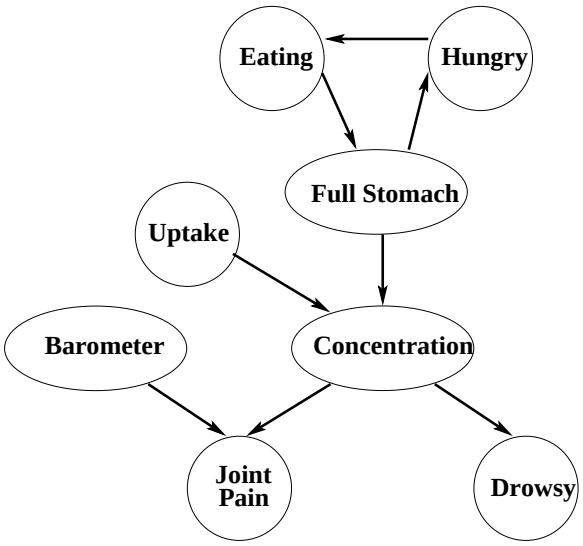


Figure 4.6: Drug Effect Network

shown in Figure 4.6. At $t = 0$ the person is not hungry, is not eating, has an empty stomach and is not drowsy. He has joint pain due to the falling barometric pressure and takes the drug to alleviate the pain.

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

More specifically, the intensity matrix of X_0 is

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

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

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

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

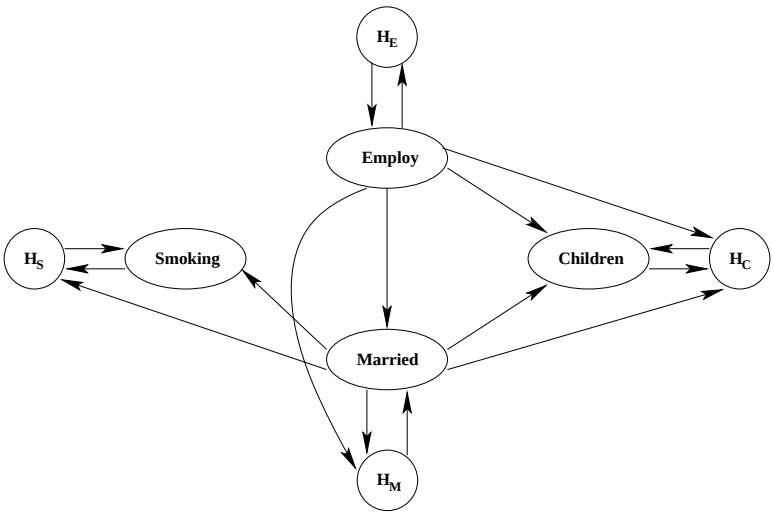


Figure 4.7: British Household Panel Survey Network

histories. We applied the structural EM algorithm in Nodelman et al. [2005b] and learned the structure of the network shown in Figure 4.7. We then estimated the parameters of the network using the EM algorithm and exact inference. We consider the learned model as the true BHPS network model for these experiments.

4.6.2 Evaluation Method

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

In the inference task, each evidence is a partially observed trajectory of the CTBN network. The evidence is generated using two methods. The first method is to set it manually. The second is to generate a trajectory using the forward sampling algorithm and randomly remove some parts of the sampled trajectory. For the latter, we repeated the following proce-

dure n times: for each variable, we randomly removed the information of the trajectory from t_s to $t_s + \gamma T$, where T is the total length of the trajectory, t_s is randomly sampled from the $[0, T - \gamma T]$ uniform distribution and $\gamma < 1$. After we run the removing procedure n times, there are at most $n\gamma$ duration of information missing for each variable. In all comparisons, this procedure was applied once and the same evidence was given to all algorithms.

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

For each query, we ran the sampling algorithms with different sample sizes, M. For each sample size, we ran the experiment N times. We calculated our query according to Equation 4.2 and compared the result to the true value calculated using exact inference. We used two metrics: the relative bias $\frac{|\sum v_M - v^*|}{v^* N}$, where v_M is the query value of sampling algorithm with sample size M, and v^* is the true value; and the relative standard deviation $\frac{\sigma_M}{v^*}$ where σ_M is the standard deviation from the true value when sample size is M. For each sample size, we also recorded the average running time $\bar{t}_M$ of each experiment and used $\bar{t}_M$ to evaluate the efficiency of the algorithm.

In the learning task, we used the sampling algorithms to estimate the parameters of a CTBN network given some partially observed data. Monte Carlo EM [Wei and Tanner, 1990] was applied in this task: In each iteration, we used the sampling based algorithm to estimate the expected sufficient statistics given the incomplete data and used Equation 3.6 to compute the parameters.

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

4.6.3 Experimental Results: Inference Task

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

Comparison of Importance Sampling and Predictive Lookahead

We first tested the importance sampling algorithm and the predictive lookahead modification using the drug effect network. We set the observed evidence: on $t = [0, 1)$ the stomach is empty, on $t = [0.5, 1.2)$ the barometer is falling, and on $t = [1.5, 2.5)$ he is drowsy. Our query is the expected total amount of time that he has no joint pain on $[0, 2.5)$. (The true value is 0.1093). We ran the two algorithms with sample sizes, M, from 5 to 90000. For each sample size, we ran the algorithms $N = 1000$ times.

The results are shown in Figure 4.8. Both algorithms achieve the correct result when the sample size is large. The standard deviation decreases at a rate of $O(\frac{1}{\sqrt{M}})$ (shown by the thin solid line). The sampling algorithm with prediction achieves lower standard deviation than the non-prediction version.

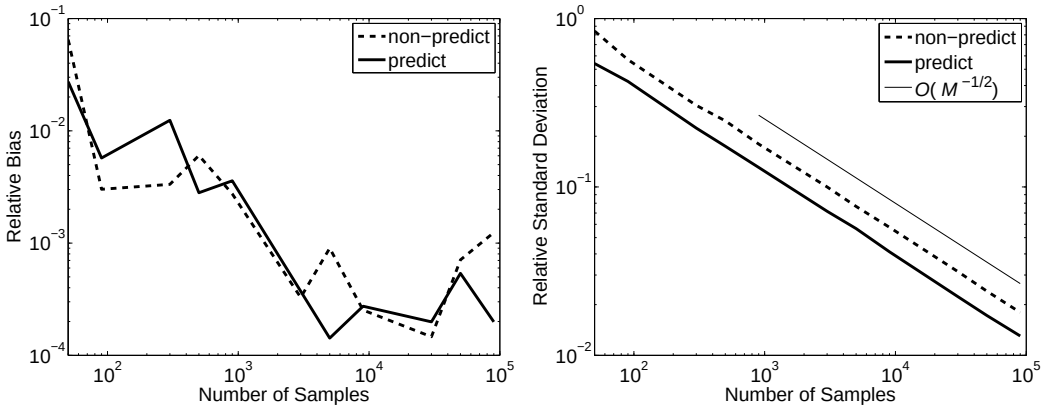
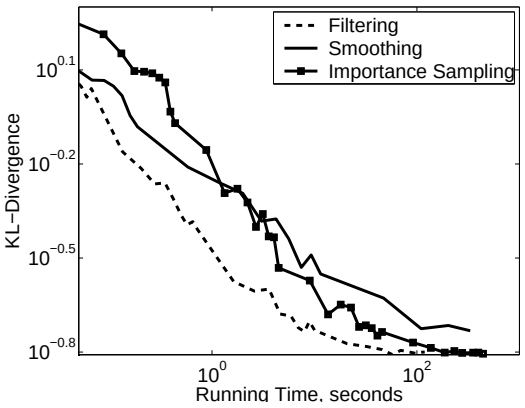


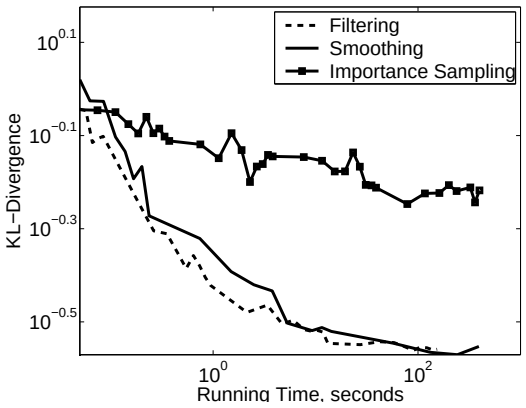
Figure 4.8: Relative bias and standard deviation of sampling with and without predictive lookahead.

Importance Sampling, Particle Filtering and Smoothing

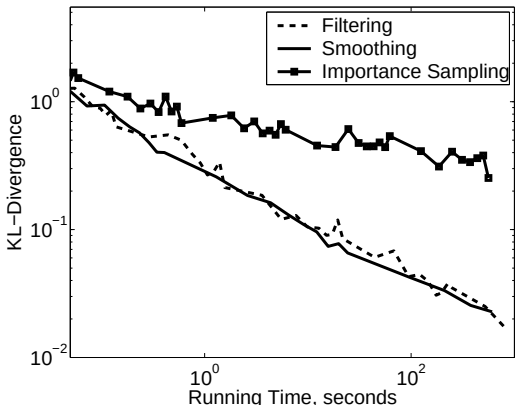
We then used the chain network to evaluate the efficiency of the importance sampling, particle filtering, and smoothing algorithms. We assumed that only X_4 was observed in this experiment. We used four different evidences. The first one is a simple evidence: only part of the behavior of X_4 is observed: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



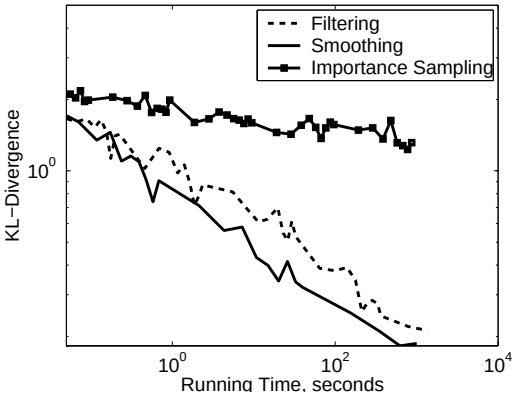
(a) Simple evidence ($T = 3$)



(b) Complex evidence ($T = 3$)



(c) Complex evidence ($T = 6$)



(d) Complex evidence ($T = 9$)

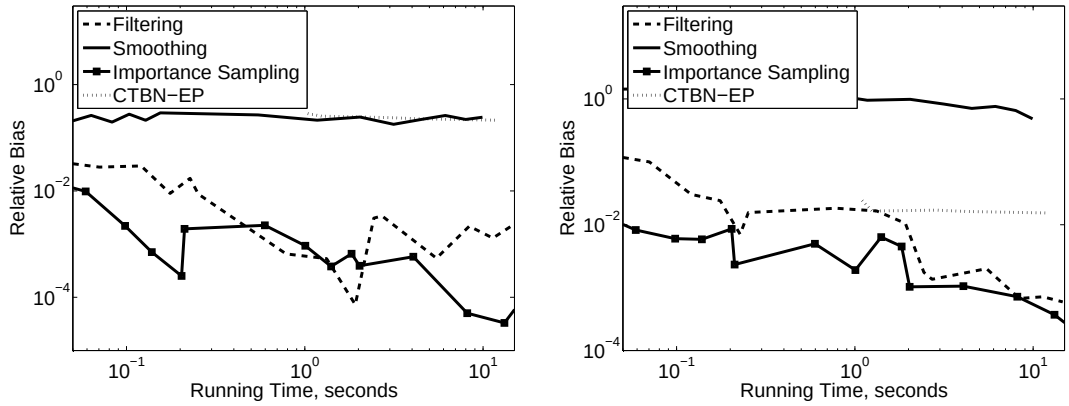
Figure 4.9: Time-efficiency comparison of particle filtering, smoothing and importance sampling

Figure 4.9 shows the efficiency of the three algorithms. In Figure 4.9(a), we used the simple evidence. In Figure 4.9 (b)-(d), we used the evidence with X_4 fully observed and $T = 3, 6, 9$ respectively. In all four cases, the particle filtering and smoothing algorithms both outperform the importance sampling algorithm when the sample size is small (small running time). For simple evidence (Figure 4.9(a)), the importance sampling algorithm achieves comparable performance when the sample size is large. When the evidence is complicated (Figure 4.9 (b)-(d)), the error of importance sampling is large, even we use very large sample sizes. When the trajectory is short, the particle filtering algorithm is slightly better than the particle smoothing algorithm. This is because the filtering algorithm can generate more samples than the smoothing algorithm with the same running time. However, as the trajectory length increases, the particle smoothing algorithm outperforms the filtering algorithm due to particle diversity problems.

Comparison of Importance Sampling and EP

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

We used the same evidence as in Section 4.6.3 on the drug effect network and answered two queries: the total amount of time that the concentration is low and the total amount of



(a) Concentration (b) Joint pain

Figure 4.10: Comparison to expectation propagation: Drug Network

time the person has no joint pain. For the EP algorithm, we first tried segmentations that were split at the evidence. We then gradually decreased the time interval of the segments to 0.15. The results of accuracy with respect to running time are shown in Figure 4.10. The importance sampling algorithm and the particle filtering algorithm outperforms the EP algorithm in answering both queries. Among the sampling-based algorithms, the importance sampling algorithm performs the best and the smoothing algorithm is the worst. This is not surprising given that most of the nodes are binary. At each transition time, the sampled trajectory has no choice as to the next state. Therefore, smoothing (or filtering) has less effect as there is no need to intelligently select the next state. However, the extra computation time for resampling and backward simulation makes the filtering and smoothing algorithm less efficient.

As mentioned above, we did not employ the adaptive splitting method of Saria et al. [2007]. It would not have changed our results much. The left-most points in Figure 4.10

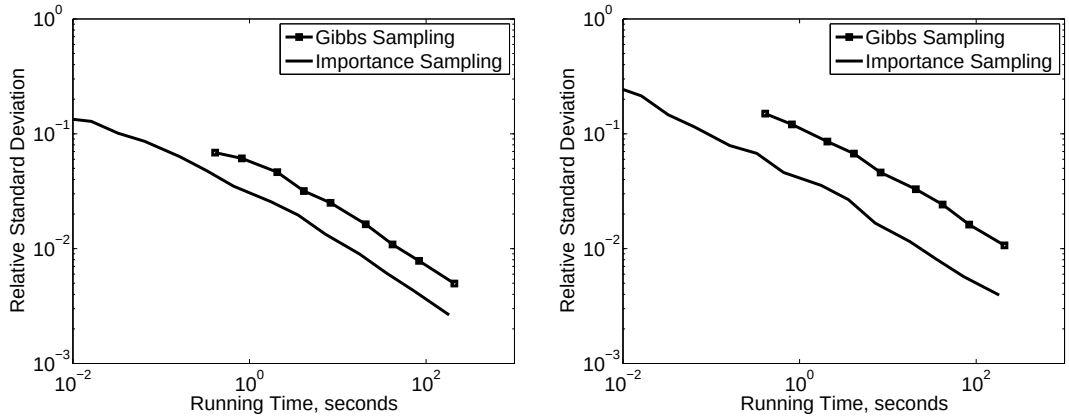
correspond to the minimum number of splits. (They are as fast as possible.) The right-most points of the Figure 4.10 correspond to many fine splits, and are about as accurate as possible, and we can see that the accuracy has flattened out. So, while the horizontal widths of the EP curves would have been shortened (by allowing for the better accuracy in less time), the vertical spread would have been approximately the same. In neither plot of Figure 4.10 would this have made a large difference in the comparisons to our sampling method.

Comparison of Importance Sampling and Gibbs Sampling

We compared our importance sampling algorithm to the Gibbs sampling algorithm as discussed in El-Hay et al. [2008]. We used three CTBN network models: the drug effect network, the BHPS network, and the chain structured network.

For each network, we randomly generated evidence using the procedure described in Section 4.6.2. We set $n = 4$ and $\gamma = 0.2$. Thus, at most 80% of the information is missing for each variable.

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



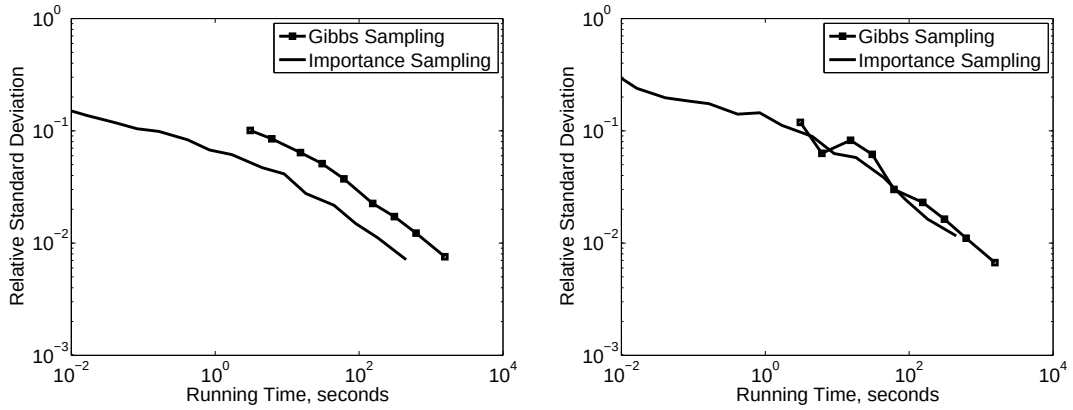
(a) Query of Time (b) Query of Transition

Figure 4.11: Comparison to Gibbs sampling: drug network. Note burn-in time for Gibbs Sampling is not included (3.94 seconds on average).

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

Using enough running time (sample size), we observed that both algorithms could answer the queries accurately (with a relative bias below 0.1%). The decreasing of the relative standard deviation with respect to the running time of the two algorithms are shown in Figure 4.11. The average "burn-in" time for the Gibbs sampler is about 3.94 seconds. From the figure, we can see that importance sampling outperforms the Gibbs sampling in answering both queries.

For the BHPS network, we set the evidence from $t = 0$ to $t = 50$ (years). We asked similar queries: the expected total amount of time a person's employment status is as a student and the expected number of times that he becomes employed. We chose the same



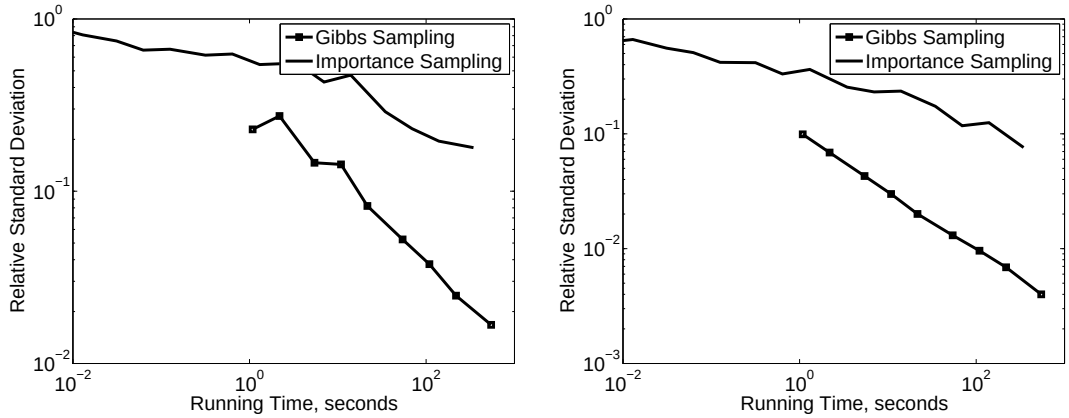
(a) Query of Time (b) Query of Transition

Figure 4.12: Comparison to Gibbs Sampling: BHPS network. Note burn-in time for Gibbs Sampling is not included (30.88 seconds on average).

sample sizes as on drug effect network and ran each sample size $N = 100$ times. Figure 4.12 shows the result of the decreasing of the standard deviation of the two algorithms. The average "burn-in" time for Gibbs sampling algorithm in this experiment is 30.88 seconds.

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

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



(a) Query of Time (b) Query of Transition

Figure 4.13: Comparison to Gibbs sampling: chain network. Note burn-in time for Gibbs Sampling is not included (11.42 seconds on average).

cumulative distribution function. Using the same amount of time, importance sampling can sample far more trajectories, which outperforms Gibbs sampling.

We last compared these two algorithms using the chain network. The evidence trajectory begins at time $t = 0$ and ends at time $t = 5$. We set the queries to be the expected total amount of time X_2 stays in state s_1 and the expected number of times that X_2 transitions from s_0 to s_1. Figure 4.13 shows the result over $N = 100$ runs. The average "burn-in" time for the Gibbs sampling algorithm in this experiment is 11.42 seconds.

Gibbs sampling achieved a better performance in this experiments. The result is not surprising. As we have mentioned before, the chain structured network is nearly deterministic, and it is the hardest case for the importance sampling algorithm. We further examined the randomly generated evidence. The only observed state on X_0 is s_0, which makes this experiment even harder for the importance sampling algorithm. However, it is a very easy case for the Gibbs sampling algorithm since it is nearly deterministic and is structurally simple.

(There are only at most one parent and one child for each node.) Although importance sampling can generate many more samples in the same period of time, most of these samples are trajectories with very small weights.

4.6.4 Task in Parametric Estimation

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

We used the drug effect network for this experiment. We sampled increasing numbers of trajectories of 5 time lengths. To hide part of the trajectory, we did the following: In each iteration, for each variable we randomly selected a time window of 0.5 time lengths and removed the content in that window. We repeated this until we dropped 50% of the content of the trajectory. We used these incomplete trajectories as our training data. We sampled another 200 trajectories with the same length to be our testing data.

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

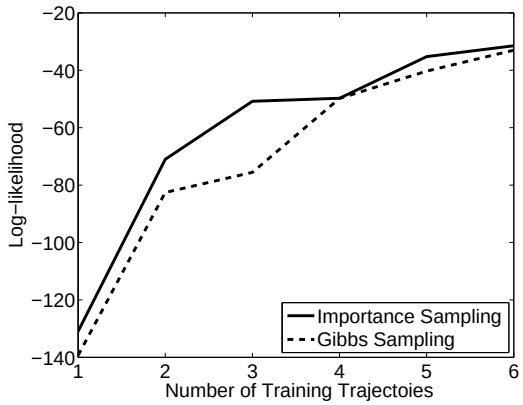


Figure 4.14: Learning results for drug effect network

We chose the initial parameters for the EM algorithm by sampling the diagonal elements of the conditional intensity matrices from the Gamma distribution with parameters $(0.5, 1)$ and sampling the transition probabilities from a Dirichlet distribution. We randomly sampled 5 models as the initial parameters for the EM algorithm. For each initial parameter set, we ran the EM algorithm 10 times. We evaluated the learning accuracy by calculating the average log-likelihood of the testing data on the 50 learned networks. To compare the running efficiency of the two sampling-based algorithm, we fixed the total amount of time for the sampler to generate samples in each EM iteration. For the Gibbs sampling algorithm, we dropped the first 50 trajectories as "burn-in" iterations. Figure 4.14 shows the results as we increased the number of training trajectories from 1 to 6. The amount of time for sampling in each EM iteration was set to be 8 seconds. The number of EM iterations was fixed to be 15.

Both algorithms obtain higher log-likelihood on the testing data when we increase the number of training trajectories. The importance sampling algorithm achieved better estima-

tion than the Gibbs sampling algorithm, especially when the number of training data was small.

Discussion

From the results, we can see that our importance sampling algorithm performs well in both inference and learning tasks. When the sample size is large, our algorithm achieves the correct value in the inference experiments. As for the efficiency of the algorithm, our importance sampling outperforms both EP and Gibbs Sampling in most of the experiments.

Our importance sampling algorithm did not perform very well in the experiment with the chain network. As all the variables are highly correlated in the chain network, choosing a proposal distribution that only consider the upcoming of each single sampled variable is insufficient. An alternative solution is to use MCMC methods. However, in our experiments, we found that Gibbs sampling is very computationally expensive. Thus, in real world applications such as social networks, where network size is usually very large and the network structure is complex, we use Metropolis-Hastings algorithm to lower the computation cost. We will discuss this in Chapter 6.

4.7 Conclusion

We have presented an approximate inference algorithm with two variations based on importance sampling. We naturally extended the algorithm to sequential Monte Carlo methods such as particle filtering and smoothing in CTBNs. A Metropolis-Hastings algorithms for

CTBNs was also developed based on the importance sampling algorithm. We applied our sampling algorithm to synthetic networks and a network derived from real data. We evaluated the efficiency of our algorithms and compared to other approximate inference algorithms based on expectation propagation and Gibbs sampling. Our importance sampling algorithm outperforms both in most of the experiments.

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

Chapter 5

Continuous-Time Social Network Dynamic Model

In the previous two chapters, we introduced the CTBN model and our sampling-based approximate inference algorithms for CTBNs using importance sampling. CTBNs provide a representation language to model large dynamic systems compactly and to explore dependencies among variables in the systems. Our importance sampling algorithms allow us to perform inference in a dynamic system even when it is large and complicated. Social networks are one such important type of dynamic systems in our daily life. In this chapter, we first review several sociology models for social network dynamics. In the next chapter, we then demonstrate that these models can be viewed as CTBNs and our importance sampling algorithms can be used as the basis to estimate the parameters of these models.

5.1 Background

A social network models the relationships (such as friendship or co-authorship) among actors (such as individuals or companies). The relationships among actors can be represented using a network with N nodes, each representing an actor, where N is number of actors. An arc from node i to node j represents a tie or relationship from actor i to actor j. At any time, the status of the tie from an actor i to another actor j can be represented using a binary variable Y_{ij} with $Val(Y_{ij}) = \{0, 1\}$. Therefore, the structure of the entire network can be described using an $n \times n$ *adjacency matrix* $Y = (Y_{ij})$. The dependencies among the actors in a social network are complex. Understanding these dependencies can help us in many areas. For example, when recommending a new game to a user on Facebook, it is very useful to know whether the person's friends are also interested in the game. It can also help us control the spread of a disease or predict the reactions of terrorists. Therefore, there has been a surge of studies on analyzing social networks in recent years.

5.2 Social Network Dynamic Models

Usually, social networks are not static but evolve in continuous time. The dynamic information is very important when modeling social networks. However, much of the past work focuses on building probabilistic models of social networks using static network snapshot data.

5.2.1 Static Social Network Models

One of the widely used models is the exponential random graph family, which was first proposed as the p_1 model [Holland and Leinhardt, 1981] and was further developed into the exponential random graph model (ERGM, or p^* model) [Anderson et al., 1999]. The ERGM defines the logarithm of the probability of a complete social network structure to be a linear combination of some structural features of the network. Generalizations of ERGM called curved exponential family models are presented in Hunter and Handcock [2006] which use non-linear parametrizations of the network structure probabilities.

Another popular model is the latent space model [Hoff et al., 2002], which assumes that each actor has a position in an unobserved "social space." The probability of the presence or absence of a link between two actors depends on their positions in the latent space. It also assumes that the probability of the link is conditionally independent of all the other links in the system given the unobserved positions of the two actors. Inference is performed using maximum likelihood and a Bayesian framework.

These works described above only focus on the static properties of social networks. Social networks, however, always evolve over time. It is therefore important to study the dynamics of social networks. Indeed, more and more work emphasises the evolution of social networks.

5.2.2 Discrete-time Model

Some work extends the previous static social network models to dynamic social network models using discrete-time models.

A class of hidden temporal exponential random graph models (htERGMs) was proposed by Guo et al. [2007], which is an extension of ERGMs for modeling networks evolving over discrete time steps. The network topology is considered to be a latent process and is assumed to be conditionally independent given the topology of the previous time step. The observed attributes of all the nodes of the network at time step t are conditionally independent of all others given the network topology and some time-invariant global features. Learning and inference on htEMGRs can be performed using sampling algorithms.

Sarkar and Moore [2005] extended the latent space model of Hoff et al. [2002] into a dynamic system that captures the changing of friendships, which are similar to hidden Markov chains. It assumes that the position of each individual can move in discrete time steps according to a transition model. The model further makes a standard Markov assumption that the position of an actor in the latent space at time t is conditionally independent of all past positions given the position at the previous time step. The observed graph at t is conditionally independent of all other positions and graphs given the positions at t. Inference can be performed by running the standard forward-backward algorithm on the model.

Liben-Nowell and Kleinberg [2003] provided a link prediction method that tries to predict new links of the network based purely on the social network structure. Given a snapshot of a social network, the prediction method calculates a connection weight score of each pair

of nodes in the network. A new link is predicted between the pair of nodes with the highest score. The score is based on the structure of the observed network such as the shortest path distance of the two nodes or the number of neighbors shared by the two nodes.

These discrete-time models capture the dynamics of social networks and are able to make predictions on the changing of the network. However, social networks always evolve asynchronously since there is no global coordination of the actors. As we discussed in previous chapters, discrete-time models have several limitations when handling these asynchronous dynamic systems. In many cases, a continuous-time model can provide more flexibility in modeling the dynamics of social networks.

5.2.3 Continuous-time Models

The idea of using continuous-time model is not new. Early models such as the reciprocity model can be found in Wasserman [1979]. The reciprocity model considers the link pair $(Y_{ij}$,$Y_{ji})$ between any two actors i and j in a network as a homogeneous continuous-time Markov process with four states (00, 01, 10, 11). It further makes the dyad independence assumption that each pair of (Y_{ij}, Y_{ji}) is independent of all the other links in the network. Therefore, the network as a whole is also a homogeneous continuous-time Markov process which includes many independent processes.

Another early model is the popularity model in Wasserman [1980]. It also models each link pair (Y_{ij}, Y_{ji}) as an independent continuous-time Markov process. However, the transition intensity of each Markov process depends on the number of incoming links of the actor

that the transition link points to. Thus, the intensity of each Markov process is time variant and the state of popular actors who have more incoming links may change faster.

These models are computationally efficient because of the dyad independence assumption. But the assumption also limits the ability of the model to represent some common properties of the social network, such as transitivity. (Actor i has higher probability of creating a friendship with actor j if i's friends also know j.)

Usually, the evolution of each link between any two actors also depends on the link status of other actors. Therefore, a more general model for the dynamics of social networks should allow link change probabilities to also depend on the entire network structure. The actor-oriented model [Snijders, 2005], an extension of the reciprocity model, is such a model. In this model, the evolution of the entire network is still modeled as a homogeneous continuous-time Markov process. However each link variable is modeled as an inhomogeneous Markov process. The probability that a link changes can depend on the entire network structure. The evolution of the network is modeled as the actors making decisions to add or remove links to maximize a utility function. The model can be simulated using the forward sampling method, and the method of moments [Bowman and Shenton, 1985] is used to estimate parameters. An alternative Bayesian based parameter estimation method is implemented in Koskinen and Snijders [2007].

One important factor we should consider is the characteristic attributes of actors. The characteristic attributes of the actors and the network's structure (both time-variant) may depend on each other. For example, people who have the same interests are likely to be-

come friends and friends are likely to influence each other's interests. Such effects should be considered when modeling the social networks. Snijders et al. [2007] extended the actor-oriented model to the network-attribute co-evolution model which added effects between the network structure and the actors' attributes. Steglich et al. [2006] showed an application of this model. They studied the dynamics of a friendship network considering attributes of actors, such as alcohol consumption. We will give detailed introduction of the network-attribute co-evolution model in the following sections.

5.3 Network-attribute Co-evolution Model

There are two types of variables in the network-attribute co-evolution model: the evolving pair-wise relationships among the N actors and the $H \geq 1$ discrete-valued attributes[1] of each actor. The number of actors is fixed during the entire process. At any time t, the ties can be described as a directed graph, which is represented by an $N \times N$ adjacency matrix $Y(t)$, where $Y_{ij}(t)$ represents the relation directed from actor i to actor j ($i, j \in \{1, \ldots, N\}$). $Y_{ij}(t) = 1$ if there is a tie from actor i to j and $Y_{ij}(t) = 0$ otherwise. Self relations are not considered in the network. The actors' attributes at t can be represented by H integer vectors $Z_h(t)$ of size N, where $Z_{hi}(t)$ denotes the value of actor i on attribute h. Therefore, the network-attribute co-evolution is modeled using the stochastic process $X(t) = (Y(t), Z_1(t), \ldots, Z_H(t))$.

[1] Snijders et al. [2007] call the attributes of the actors "behaviors." We call them "attributes" to avoid confusion with the dynamic behavior of the model.

The network-attribute co-evolution model assumes that the process $X(t)$ is a continuous-time Markov process. The evolution of the network is modeled as actors making decisions to maximize their satisfaction with the network: an actor may choose to add or remove an outgoing tie, or change the value of one attribute in order to (approximately and locally) maximize a utility function. At any time t, given the current state $X(t)$, the decisions made by the actors are conditionally independent.

It is further assumed that when making a decision, the actor can only change one outgoing tie or change one attribute value. Because of the continuous-time nature, no two events may occur at exactly the same time. Thus, at any time, only one actor can add or remove an outgoing tie or increase or decrease one value unit of an attribute. For each actor i, the times between two network changes and between two attribute decisions are exponentially distributed with parameters λ_i^n and λ_i^a, called rate functions. Usually, they are assumed to be constant values.

At a transition point, the actor chooses to add or remove a tie to maximize the value of the network utility function $f_i^n(\beta^n, \boldsymbol{y}\langle i,j\rangle, \boldsymbol{z}) + \epsilon^n$, or to change the value of an attribute to maximize the attribute utility function $f_i^a(\beta^a, \boldsymbol{y}, \boldsymbol{z}_h\langle i,\delta\rangle) + \epsilon^a$, where $f_i^n(\beta^n, \boldsymbol{y}\langle i,j\rangle, \boldsymbol{z})$ and $f_i^a(\beta^a, \boldsymbol{y}, \boldsymbol{z}_h\langle i,\delta\rangle)$ are the objective functions for network and attribute decisions respectively. $\boldsymbol{y}$ and $\boldsymbol{z}$ are the current states of the network and attributes. $\boldsymbol{y}\langle i,j\rangle$ denotes the state of the network after the tie from i to j changes. $\boldsymbol{z}_h\langle i,\delta\rangle$ denotes the state of the attribute after actor i changes the attribute $\boldsymbol{z}_h$ by δ, where $\delta \in \{-1,+1\}$. Both objective functions are modeled as a weighted sum of effects (features) that depend on the topology of the network

and the attribute values. The functions have the form

$$f_i(\beta, \boldsymbol{y}, \boldsymbol{z}) = \sum_{k=1}^{L} \beta_k s_{ik}(\boldsymbol{y}, \boldsymbol{z})$$

where $s_{ik}(\boldsymbol{y}, \boldsymbol{z})$ is an effect that expresses a property of the network structure and the attribute values from the view of actor i, and L is the number of effects the actor considers.

Example 5.3.1 *We can define the utility functions as follows. For the network utility function, assume that the evolution of the network structure depends on two effects $s^n_{i1}(\boldsymbol{y}, \boldsymbol{z})$ and $s^n_{i2}(\boldsymbol{y}, \boldsymbol{z})$. Let $s^n_{i1}(\boldsymbol{y}, \boldsymbol{z})$ be the "density effect," defined as the number of out-going ties from i. Let $s^n_{i2}(\boldsymbol{y}, \boldsymbol{z})$ be the "reciprocity effect," defined as the number of reciprocated links to i. For the attribute utility function, we assume it depends on the "attribute tendency" and the "similarity effect" of actor i. The "attribute tendency" is defined as the current attribute value of i. The "similarity effect" is defined by a similarity function. Then the utility function $f^n_i(\beta^n, \boldsymbol{y}\langle i, j\rangle, \boldsymbol{z})$ and $f^a_i(\beta^a, \boldsymbol{y}, \boldsymbol{z}_h\langle i, \delta\rangle)$ are*

$$f^n_i(\beta^n, \boldsymbol{y}\langle i, j\rangle, \boldsymbol{z}) = \beta^n_1 \sum_j y_{ij} + \beta^n_2 \sum_j y_{ij} y_{ji}$$

$$f^a_i(\beta^a, \boldsymbol{y}, \boldsymbol{z}_h\langle i, \delta\rangle) = \beta^a_1 z_{ih} + \beta^a_2 \sum\nolimits_j y_{ij} \mathbf{sim}_{ij}$$

where $\mathbf{sim}_{ij} = 1 - |z_{ih} - z_{jh}|/Range(z_h)$ is the attribute similarity between actors i and j.

If $\beta^n_1 = -2$, $beta^n_2 = 1.5$, $\beta^a_1 = 0.5$, and $\beta^a_2 = 1$, this means that when making decision to change the status of a link, actor i is unlikely to randomly create a relation ($\beta^n_1 < 0$, discour-

aging dense network) and is likely to create reciprocated connections ($\beta_2^n > 0$, encouraging reciprocated links). The actor has a preference for high value of attribute Z_h ($\beta_1^a > 0$, encouraging higher value of z_h), and the actor tends to change the attribute to be similar to his friends ($\beta_2^a > 0$, encouraging similarity).

ϵ^n and ϵ^a are random noises. Following Snijders et al. [2007], they are set to be Gumbel distributions with mean 0 and scale parameter 1. Then the transition probability of actor i changing the tie to j and actor i changing the value of z_h by δ are

$$P(\boldsymbol{y}\langle i,j\rangle|\boldsymbol{y},\boldsymbol{z}) = \frac{\exp(f_i^n(\beta^n,\boldsymbol{y}\langle i,j\rangle,\boldsymbol{z}))}{\sum_{k\neq i}\exp(f_i^n(\beta^n,\boldsymbol{y}\langle i,k\rangle,\boldsymbol{z}))} \tag{5.1}$$

$$P(\boldsymbol{z}_h\langle i,\delta\rangle|\boldsymbol{y},\boldsymbol{z}) = \frac{\exp(f_i^a(\beta^a,\boldsymbol{y},\boldsymbol{z}_h\langle i,\delta\rangle))}{\sum_{\delta}\exp(f_i^a(\beta^a,\boldsymbol{y},\boldsymbol{z}_h\langle i,\delta\rangle))}\ . \tag{5.2}$$

Given the rate functions and the transition probabilities, the intensity matrix of the continuous Markov process $X(t)$ can be written as

$$q_{\boldsymbol{x},\hat{\boldsymbol{x}}} = \begin{cases} \lambda_i^n P(\boldsymbol{y}\langle i,j\rangle|\boldsymbol{y},\boldsymbol{z}) & \text{if } \hat{\boldsymbol{x}} = (\boldsymbol{y}\langle i,j\rangle,\boldsymbol{z}) \\ \lambda_i^a P(\boldsymbol{z}_h\langle i,\delta\rangle|\boldsymbol{y},\boldsymbol{z}) & \text{if } \hat{\boldsymbol{x}} = (\boldsymbol{y},\boldsymbol{z}_h\langle i,\delta\rangle) \\ -\sum_i(\lambda_i^n+\lambda_i^a) & \text{if } \hat{\boldsymbol{x}} = \boldsymbol{x} \\ 0 & \text{otherwise.} \end{cases} \tag{5.3}$$

When there are no observations, we can generate trajectories for $X(t)$ using standard forward sampling method for continuous-time Markov processes.

Since the network-attribute model is an extension of the actor-orientated model, the actor-oriented model can be considered as a special case of the network-attribute model where the evolution of a social network only depends on the structure of the network and no actor attributes are considered. Thus, we only need the rate function λ_i^n and utility function $f_i^n(\beta^n, \boldsymbol{y}\langle i,j\rangle, \boldsymbol{z})$ to model the dynamics of a social network under the actor-oriented model. The dynamics of the entire social network are still a continuous-time Markov process. The intensity matrix of the process is

$$q_{\boldsymbol{x},\hat{\boldsymbol{x}}} = \begin{cases} \lambda_i^n P(\boldsymbol{y}\langle i,j\rangle|\boldsymbol{y}) & \text{if } \hat{\boldsymbol{x}} = \boldsymbol{y}\langle i,j\rangle \\ -\sum_i \lambda_i^n & \text{if } \hat{\boldsymbol{x}} = \boldsymbol{x} \\ 0 & \text{otherwise.} \end{cases} \tag{5.4}$$

5.4 Parameter Estimation

Snijders et al. [2007] assume that observations are only available at discrete time points $t_1 < t_2 < \cdots < t_M$, where $M \geq 3$. The parameters $\alpha = (\lambda^n, \lambda^a, \beta^n, \beta^a)$ are estimated based on M network observations $y(t_1), \ldots, y(t_M)$ and attribute observations $z(t_1), \ldots, z(t_M)$. Parameters are estimated using the method of moments (MoM) [Bowman and Shenton, 1985].

MoM estimates the parameters such that the expected values of some statistics $D(\boldsymbol{y}, \boldsymbol{z})$ under the estimated parameter are equal to the observed values. The statistics used by Snijders et al. [2007] for each parameter are as follows.

Parameter	$D(\boldsymbol{y}, \boldsymbol{z})$
λ^n	$\sum_{i,j} \lvert y_{ij}(t_{m-1}) - y_{ij}(t_m) \rvert$
λ^a	$\sum_{h,i} \lvert z_{hi}(t_{m-1}) - z_{hi}(t_m) \rvert$
β_k^n	$\sum_i s_{ik}^n(\boldsymbol{y}(t_m), \boldsymbol{z}(t_{m-1}))$
β_k^a	$\sum_i s_{ik}^a(\boldsymbol{y}(t_{m-1}), \boldsymbol{z}(t_m))$

Their estimation algorithm uses the Newton-Raphson method starting with random parameters. In each iteration, parameters are updated so that the expected statistics $D(\boldsymbol{y}, \boldsymbol{z})$ listed above equal the statistics of observation data. The expectation of the statistic values are calculated using forward sampling between two consecutive time points t_m and t_{m+1} without considering the observations.

MoM estimates the parameters using some statistics of the observations data, which implies that there should not be missing values in the M network observations $y(t_1), \ldots, y(t_M)$ and attribute observations $z(t_1), \ldots, z(t_M)$. However, in real applications, missing values in observation data are inevitable. In Snijders [2005], network observations are completed by simply replacing missing values with zeros. Huisman and Steglich [2008] only set missing values to be zero in the first network observation so that forward sampling algorithm can be performed. To estimate parameters, only observations that are available at two consecutive time points are used, which means that a lot of useful information is discarded.

5.5 Summary

Social networks usually evolve in continuous time. The actor-oriented model and the network-attribute co-evolution model provide approaches to model the dynamics of social networks naturally using continuous-time Markov processes. Also, the evolution of social networks is allowed to depend on the entire network structure. Furthermore, the network-attribute co-evolution model includes the influence of the dynamic attributes of actors in the model. These two models give us more flexibility when modeling the dynamics of social networks.

However, there are several limitations in these two models. Only some sufficient statistics of the evidence are considered when estimating the parameters. Samples generated to estimate the parameters do not fully agree with the evidence, which may affect the estimation accuracy. Second, fully observed network structure and attribute values at each observation point are required when calculating the sufficient statistics of observation data. However, missing values in observation data are very common in real social network applications. Finally, these two models assume that direct observations of the network structure are available at $M \geq 3$ time points. Collecting this type of social network data is very expensive, which may take several years. These facts limit the range of social network data to which these two models can be applied.

One solution for these problem is to view the actor-oriented model and the network-attribute co-evolution model as CTBNs, since they all treat the dynamic systems as continuous-time Markov processes. Algorithms in CTBNs then can be adapted to social network dy-

namic models, which we will discuss in details in the next chapter. This allows us to deal with general evidence patterns.

Chapter 6

Learning Social Network Dynamics

In the previous chapter, we introduced the actor-oriented model and the network-attribute co-evolution model. These two social network models provide us approaches to model the dynamics of social networks in continuous time, where the evolution of each tie in the network may depend on the entire structure of the network. The network-attribute co-evolution model further allows the impact of actor's characteristic attributes to be considered when studying the evolution of social networks. However, since the learning algorithm in these models only uses forward sampling, estimation accuracy may be affected. Also, only evenly-spaced, fully observed network snapshots data can be handled using the prior learning algorithms.

In this chapter, we will show that these two models can be viewed as CTBNs. Therefore, our importance sampling algorithm for CTBNs can be adapted to these social network models to develop a maximum likelihood estimation algorithm. Furthermore, using CTBN models, we present a hidden social network dynamics model in which indirect observations such as

emails events among people can be utilized. The Metropolis-Hastings algorithm developed in Chapter 4 can be applied as the basis for parameter estimation. Thus, we not only improve upon previous parameter estimation methods for social network dynamics, but also extend their range to deal with more flexible data sources.

6.1 Sampling for Learning Social Networks

The method of moments (MoM) parameter estimation method in Snijders et al. [2007] only uses some sufficient statistics of the observation data. Samples generated during the estimation procedure do not fully agree with the observations, which may affect the estimation accuracy. Additionally, calculating the true statistics, $D(\boldsymbol{y}, \boldsymbol{z})$, requires that, at each observation point, all the network and attribute values are fully observed. Thus, it is hard for MoM estimation to handle observation data with missing values or evidence about durations, common in real applications.

Since the network-attribute co-evolution model assumes that the entire state $X(t)$ of the social network is a continuous-time Markov process, we can easily convert it to a CTBN, and apply the importance sampling algorithm to the converted model. The samples generated by importance sampling algorithm are consistent with the observations, and we can apply maximum likelihood estimation.

6.1.1 Importance Sampling for Network-attribute Co-evolution Model

Given the current instantiation $(\boldsymbol{y}, \boldsymbol{z})$ of the whole system, the conditional intensity matrix $Q^n_{ij|\boldsymbol{y},\boldsymbol{z}}$ for link variable $y_{ij}(t)$ can be extracted from Equation 5.3:

$$Q^n_{ij|\boldsymbol{y},\boldsymbol{z}} = \begin{bmatrix} -\lambda^n_i P^0_{i,j} & \lambda^n_i P^0_{i,j} \\ \lambda^n_i P^1_{i,j} & -\lambda^n_i P^1_{i,j} \end{bmatrix}. \tag{6.1}$$

where $P^k_{i,j} = P(\boldsymbol{y}\langle i,j\rangle | y_{ij} = k, \boldsymbol{y}, \boldsymbol{z})$ for $k = \{0, 1\}$. The conditional intensity matrix $Q^a_{i|\boldsymbol{y},\boldsymbol{z}}$ for attribute variable $Z_{hi}(t)$ can be extracted similarly.

The transition probabilities depend on the utility functions, whose values depend on the current instantiation of the entire system. Independencies among variables only hold for particular assignments to certain other variables (analogous to context sensitive independence for Bayesian networks). For example, if the network utility function's features include the attribute similarity to connected actors, then the dynamics of link Y_{ij} can potentially depend on all Z_k, but at any given instant, only depends on those Z_k for which $Y_{ik} = 1$. If any one of these links Y_{ik} changes, the independencies between Y_{ij} and $\{Z_k\}$ will change. However, the independencies are fixed between any two consecutive transitions. Therefore, this dynamic CTBN structure prevents efficient use of other approximate CTBN inference algorithms like expectation propagation [Saria et al., 2007]. We could apply the Gibbs sampling algorithm of El-Hay et al. [2008], but it must calculate the true posterior density between every two con-

secutive transitions which can be arbitrarily complex. Sampling from the posterior density involves binary search. As we have shown in Chapter 4, it is very computational costly.

Using the converted CTBN model, we can apply the importance sampling algorithm for the model. This also allows for general evidence patterns beyond complete snapshots.

6.1.2 Maximum Likelihood Estimation

The importance sampling algorithm can generate weighted samples that fully agree with the observations. The log-likelihood of the samples can be calculated using Equation 3.1 and Equation 4.3. Therefore, we can use maximum likelihood estimation to learn the parameters $\alpha = (\lambda^n, \lambda^a, \beta^n, \beta^a)$. Since the data are partially observed, we employ the Monte Carlo expectation maximization (MCEM) algorithm [Wei and Tanner, 1990]. For this application of EM, the steps are as follows.

Expectation Step: Using the current parameters, apply the importance sampling algorithm to generate m weighted samples. Calculate the sufficient statistics and the log-likelihood of the samples.

Maximization Step: Update parameters, using the sufficient statistics and log-likelihood as if they came from the complete data. Rate parameters λ_i^n and λ_i^a are set to be $M^n[i]/T$ and $M^a[i]/T$ respectively, where $M^n[i]$ and $M^a[i]$ are the number of link changes and attribute changes for actor i, respectively. We use conjugate gradient ascent [Press et al., 1992] to estimate the weight parameters β^n and β^a since they cannot be solved analytically from the log-likelihood function.

Notice that the rate parameters and the weight parameters can be updated separately. To increase the accuracy, we divide the EM iterations into two loops so that the two sets of parameters are estimated alternatingly. Specifically, in each EM iteration, we first fix the current parameters for the weight parameters, estimate the rate parameters by conducting the above two steps. We then update the rate parameters, fix them, and estimate the weight parameters using the above two steps.

The rate parameters are calculated using the expected number of transitions of the model. Since the time intervals between transitions are sampled from the exponential distribution with the current intensity rate q, it is difficult to sample a trajectory with a slower rate. (The algorithm tends to add additional transitions to get the trajectory to agree with the evidence.) Therefore, if the initial rate parameter is larger than the true value, it is unlikely that the EM algorithm will converge to the true rate with a reasonable number of samples. To avoid this, instead of sampling transitions using the current intensity q, we sample transitions from the exponential and truncated exponential distribution with intensity $q/2$. We adjust the sample weights accordingly.

6.2 Hidden Social Network Dynamics Model

The network-attribute co-evolution model assumes that the adjacency matrix $Y(t)$ can be observed $M \geq 3$ times. Observation of network structure for each actor are collected using survey or interview instruments, which sometimes takes several years. It is very expensive

even obtaining one complete observation of the network. However, communication events among people, such as emails, instant messages, and phone calls, are easier to collect. We can fully observe these events continuously with lower cost. The occurrence of such an event depends on the connection status between the actors involved. Thus, although they may not be as accurate, these events indirectly reflect the relationships among people. In such model, the network itself is unobserved (hidden) all the time. We call this model the hidden social network dynamics model.

6.2.1 Model Definition

Let $Y(t)$ be the network of N actors, which evolves in continuous time in the same way as in the network-attribute co-evolution model. (We do not consider the attribute variables, but adding them to the model is straight-forward.) Let $O(t)$ be the observations (such as emails or phone calls) among the N actors. $O(t)$ can be fully or partially observed as $Y(t)$ evolves. $O_{ij}(t) \in O(t)$ ($i, j = 1, \ldots, N$, $i \neq j$) is the observation of communication directed from i to actor j. We assume that the dynamics of $O_{ij}(t)$ depend only on $Y_{ij}(t)$ and $Y_{ji}(t)$. Thus, the dynamics of each observation variable $O_{ij}(t)$ can be described using four conditional intensity matrices $Q^{obs}_{ij|Y_{ij}=k,Y_{ji}=l}$ ($k, l \in \{0, 1\}$), each of which corresponds to a state of $Y_{ij}(t)$ and $Y_{ji}(t)$. We further assume that all the event variables share the same set of parameters. That is, $\forall i, j$, that $i \neq j$, $Q^{obs}_{ij|Y_{ij}=k,Y_{ji}=l} = Q^{obs}_{Y_{ij}=k,Y_{ji}=l}$.

Therefore, the hidden social network dynamic model contains two sets of variables: the network variables $Y(t)$ and the observations $O(t)$. The evolution of the network variables is

the same as in the network-attribute co-evolution model. It depends on the network rate λ^n and the utility function f^n. The dynamics of each observation variable $O_{ij}(t)$ depend only on the state of link variables $Y_{ij}(t)$ and $Y_{ji}(t)$. According to the Markov properties of CTBNs, $O_{ij}(t)$ is independent of all the other variables given the entire trajectory of $Y_{ij}(t)$ and $Y_{ji}(t)$.

Example 6.2.1 *One of the commonly observed social events is mobile phone calls among people. Currently, almost everyone uses mobile phones, and mobile phone calls usually indirectly reflect the relationships among people, though perhaps not exactly. For example, not every person one calls is a friend, but most phone traffic is between friends.*

Most of the time, a mobile phone is in a standby state. When a person wants to communicate with his friend using a mobile phone, one of the the following three situations may happen.

- *The person calls his friend and has a conversation.*
- *The person sends a text message to his friend.*
- *The person calls his friend but his friend misses that call.*

Therefore, we can describe the dynamics of a mobile phone using the following conditional intensity matrix

$$Q^{obs}_{ij|Y_{ij}=k,Y_{ji}=l} = \begin{bmatrix} -q^s_{kl} & q^{call}_{kl} & q^{msg}_{kl} & q^{miss}_{kl} \\ q^c_{kl} & -q^c_{kl} & 0 & 0 \\ \infty & 0 & -\infty & 0 \\ \infty & 0 & 0 & -\infty \end{bmatrix} \begin{matrix} \textit{standby} \\ \textit{in-call} \\ \textit{text message} \\ \textit{missed call} \end{matrix} .$$

q_{kl}^{call}, q_{kl}^{msg} and q_{kl}^{miss} are the intensities of a person making a successful call, sending a message and making a no-answer call respectively. $q_{kl}^{s} = q_{kl}^{call} + q_{kl}^{msg} + q_{kl}^{miss}$ is the intensity that a mobile phone leaves the standby state. q_{kl}^{c} is the intensity of a mobile phone leaving the conversation state. When a conversation is ended, the mobile phone can only go back to the standby state. The intensities of changing from conversation state to text message state or missed call state are zeroes. Text message and missed call are all instantaneous events. There should be no duration for these two states. The corresponding intensities are set to be infinities.

6.2.2 Metropolis-Hastings Sampling Algorithm

Due to the large number of variables in the hidden social network dynamic model, exact inference is intractable. Since the model can be naturally converted to a CTBN, we can apply the importance sampling algorithm to the converted model. However, because only $Y(t)$'s children are observed, the trajectory of $Y(t)$ would be sampled blindly with no guidance from the evidence. Any incorrectly sampled variable can make the entire trajectory be highly unlikely. Given the large sample space for $Y(t)$, the importance sampler generates many trajectories with very small weights.

An alternative method is to use the Metropolis-Hastings algorithm described in Section 4.5.2. Unlike importance sampling algorithm, the Metropolis-Hastings algorithm evaluates the newly sampled trajectory σ' in each iteration. A trajectory with low probability is likely

to be rejected according to the acceptance probability $\mathcal{A}(\sigma \rightarrow \sigma')$. Therefore, the Metropolis-Hastings algorithm guides the sampling toward more likely trajectories.

Notice that in the hidden social network dynamic model, each observation variable $O_{ij}(t)$ is independent of all the other variables given the entire trajectory of $Y_{ij}(t)$ and $Y_{ji}(t)$. Given the entire trajectory of $Y_{ij}(t)$ and $Y_{ji}(t)$, queries about $O_{ij}(t)$ can be calculated using exact inference. According to the Metropolis-Hastings algorithm, the sampling procedure starts with an arbitrary trajectory when sampling $Y(t)$. Then in each iteration, the sampler randomly picks a variable $Y_{ij}(t)$ and replaces the entire trajectory of $Y_{ij}(t)$ using the importance sampling algorithm, fixing the rest of the trajectory as evidence. If the acceptance ratio of the new sampled trajectory is larger than a random number u, sampled from a $[0, 1]$ uniform distribution, the new trajectory is accepted. Otherwise, the old trajectory is kept.

Let σ be the trajectory from the previous iteration, and σ' be the sampled trajectory in the current iteration. Let P be the target distribution given by the model and P' be the proposal distribution used by the importance sampling algorithm. We need to calculate the acceptance ratio of the sampled trajectory σ' in the Metropolis-Hastings algorithm $r(\sigma, \sigma') = \frac{P(\sigma')P'(\sigma|\sigma')}{P(\sigma)P'(\sigma'|\sigma)}$.

We define σ_Y to be the trajectory of $Y(t)$ in σ, $\sigma_{Y_{ij}}$ to be the trajectory of $Y_{ij}(t)$ in σ and $\sigma_{/Y_{ij}}$ to be the trajectory of $Y(t)$ except $Y_{ij}(t)$. We define σ_O and $\sigma_{O_{ij}}$ similarly. If Y_{ij} is the

variable to be resampled, according to the Markov properties of CTBNs, we have

$$
\begin{aligned}
r(\sigma,\sigma') &= \frac{P(\sigma'_Y)P(\sigma_O|\sigma'_Y)}{P(\sigma_Y)P(\sigma_O|\sigma_Y)}\frac{P'(\sigma_Y|\sigma'_Y,\sigma_O)}{P'(\sigma'_Y|\sigma_Y,\sigma_O)} \\
&= \frac{P(\sigma'_Y)}{P(\sigma_Y)}\frac{P(\sigma_{O_{ij}},\sigma_{O_{ji}}|\sigma'_{Y_{ij}},\sigma_{Y_{ji}})}{P(\sigma_{O_{ij}},\sigma_{O_{ji}}|\sigma_{Y_{ij}},\sigma_{Y_{ji}})}\frac{\frac{P(\sigma'_{Y_{ij}}|\sigma_{/Y_{ij}})}{P'(\sigma'_{Y_{ij}}|\sigma_{/Y_{ij}})}}{\frac{P(\sigma_{Y_{ij}}|\sigma_{/Y_{ij}})}{P'(\sigma_{Y_{ij}}|\sigma_{/Y_{ij}})}}\frac{P(\sigma_{Y_{ij}}|\sigma_{/Y_{ij}})}{P(\sigma'_{Y_{ij}}|\sigma_{/Y_{ij}})} \\
&= \frac{L(\sigma'_{/Y_{ij}})}{L(\sigma_{/Y_{ij}})}\frac{P(\sigma_{O_{ij}}|\sigma'_{Y_{ij}},\sigma_{Y_{ji}})}{P(\sigma_{O_{ij}}|\sigma_{Y_{ij}},\sigma_{Y_{ji}})}\frac{P(\sigma_{O_{ji}}|\sigma'_{Y_{ij}},\sigma_{Y_{ji}})}{P(\sigma_{O_{ji}}|\sigma_{Y_{ij}},\sigma_{Y_{ji}})}\frac{w(\sigma'_{Y_{ij}})}{w(\sigma_{Y_{ij}})}
\end{aligned}
\tag{6.2}
$$

where $L(\cdot)$ is the partial likelihood function and $w(\cdot)$ is the weight contribution of the variable in the importance sampling algorithm.

If the proposal distribution is as described in Chapter 4, $w(\sigma'_{Y_{ij}})$ and $w(\sigma'_{Y_{ij}})$ in Equation 6.2 are 1 since there is no evidence on $Y_{ij}(t)$. If another proposal distribution is used, the weight contribution should be adjusted accordingly.

If the behavior of $O(t)$ is completely observed, $P(\sigma_{O_{ij}}|\sigma_{Y_{ij}},\sigma_{Y_{ji}})$ can be represented using likelihood of O_{ij} given $\sigma_{Y_{ij}}$ and $\sigma_{Y_{ji}}$. Then the acceptance ratio is

$$
r(\sigma,\sigma') = \frac{L(\sigma'_{/Y_{ij}})}{L(\sigma_{/Y_{ij}})}\frac{L(\sigma_{O_{ij}}|\sigma'_{Y_{ij}},\sigma_{Y_{ji}})}{L(\sigma_{O_{ij}}|\sigma_{Y_{ij}},\sigma_{Y_{ji}})}\frac{L(\sigma_{O_{ji}}|\sigma'_{Y_{ij}},\sigma_{Y_{ji}})}{L(\sigma_{O_{ji}}|\sigma_{Y_{ij}},\sigma_{Y_{ji}})}\frac{w(\sigma'_{Y_{ij}})}{w(\sigma_{Y_{ij}})} \tag{6.3}
$$

If $O(t)$ is partially observed, we can calculate $P(\sigma_{O_{ij}}|\sigma_{Y_{ij}},\sigma_{Y_{ji}})$ as follows. Assume that we are sampling trajectories from time 0 to time T. $O_{ij}(t)$ is observed on intervals $[t_k, t_k+\Delta t_k]$ ($k=1,\ldots,m$), where $t_{k+1} > t_k+\Delta t_k$, $t_1 \geq 0$, and $t_m+\Delta t_m \leq T$. Let $\sigma^k_{O_{ij}}$

be the observed trajectory of O_{ij} on interval $[t_k, t_k + \Delta t_k]$. Using Markov property, we have

$$P(\sigma_{O_{ij}}|\sigma_{Y_{ij}}, \sigma_{Y_{ji}}) = \prod_k \alpha_k[O_{ij}(t_k)] \times L(\sigma^k_{O_{ij}}|\sigma_{Y_{ij}}, \sigma_{Y_{ji}}) \times \beta_k[O_{ij}(t_k + \Delta t_k)]\,,$$

where $L(\sigma^k_{O_{ij}}|\sigma_{Y_{ij}}, \sigma_{Y_{ji}})$ is the partial likelihood of $\sigma^k_{O_{ij}}$, and the vectors α_k and β_k are defined component-wise as

$$\alpha_k[i] = P(O_{ij}(t_k) = i|O_{ij}(t_{k-1} + \Delta t_{k-1}), \sigma_{Y_{ij}}, \sigma_{Y_{ji}})$$

$$\beta_k[i] = P(O_{ij}(t_{k+1})|O_{ij}(t_k + \Delta t_k) = i, \sigma_{Y_{ij}}, \sigma_{Y_{ji}})\,.$$

α_k and β_k can be calculated using the standard forward-backward algorithm described in Section 3.4.2.

6.2.3 Parameter Estimation

The parameters can be estimated using the Monte Carlo EM algorithm described in Section 6.1.2. In the expectation step, we use the Metropolis-Hastings algorithm to generate equally weighted samples. In the maximization step, we update the parameters alternatingly. Calculating λ^n and β^n is the same as described in Section 6.1.2. According to Equation 3.6, the

conditional intensity matrix for observation variable $Q^{obs}_{ij|Y_{ij}=k,Y_{ji}=l}$ can be estimated as

$$q^{obs}_{o|Y_{ij}=k,Y_{ji}=l} = \frac{\sum_{i\neq j} M_{O_{ij}}[o|Y_{ij}=k, Y_{ji}=l]}{\sum_{i\neq j} T_{O_{ij}}[o|Y_{ij}=k, Y_{ji}=l]} \tag{6.4}$$

$$\theta^{obs}_{oo'|Y_{ij}=k,Y_{ji}=l} = \frac{\sum_{i\neq j} M_{O_{ij}}[o,o'|Y_{ij}=k, Y_{ji}=l]}{\sum_{i\neq j} M_{O_{ij}}[o|Y_{ij}=k, Y_{ji}=l]} \tag{6.5}$$

where $M_{O_{ij}}[oo'|Y_{ij} = k, Y_{ji} = l]$ is the number of times O_{ij} transitions from state o to o' when $Y_{ij} = k$ and $Y_{ji} = l$, $T_{O_{ij}}[o|Y_{ij} = k, Y_{ji} = l]$ is the total amount of time O_{ij} stays in state o when $Y_{ij} = k$ and $Y_{ji} = l$, $M_{O_{ij}}[o|Y_{ij} = k, Y_{ji} = l] = \sum_{o'} M_{O_{ij}}[oo'|Y_{ij} = k, Y_{ji} = l]$, and $q^{obs}_{oo'|Y_{ij}=k,Y_{ji}=l} = \theta^{obs}_{oo'|Y_{ij}=k,Y_{ji}=l} \times q^{obs}_{o|Y_{ij}=k,Y_{ji}=l}$.

When $O(t)$ is only partially observed, the expectation of the sufficient statistics above can be calculated using exact inference described in Section 3.4.2.

6.3 Experimental Results

We evaluate the learning algorithm using importance sampling on both synthetic data and real sociological data. We compare the result to the method of moments (MoM) learning algorithm. We also evaluate our learning algorithm using MCMC for the hidden social network dynamics model on a real dataset of emails.

6.3.1 Network-attribute Co-evolution Model

Synthetic Dataset

We first built a synthetic social network with 10 people and one time-variant attribute for each person. We assume the rates are homogeneous across people. We consider three effects on the network change rule (features): the density effect (number of outgoing links), the reciprocity effect (number of reciprocated links) and the attribute-related similarity. The attribute utility function considers two effects: tendency and similarity. The utility functions are therefore

$$
\begin{aligned}
f_i^n(\beta^n, \boldsymbol{y}, \boldsymbol{z}) &= \sum\nolimits_j (\beta_1^n y_{ij} + \beta_2^n y_{ij} y_{ji} + \beta_3^n y_{ij} \mathbf{sim}_{ij}) \\
f_i^a(\beta^a, \boldsymbol{y}, \boldsymbol{z}) &= \beta_1^a z_i + \beta_2^a \sum\nolimits_j y_{ij} \mathbf{sim}_{ij}
\end{aligned} \tag{6.6}
$$

where $\mathbf{sim}_{ij} = 1 - |z_i - z_j| / Range(z)$ is the attribute similarity between actors i and j. These are the same effects or features as in Snijders et al. [2007]. We set $(\beta_1^n, \beta_2^n, \beta_3^n) = (-1, 1.5, 1)$ and $(\beta_1^a, \beta_2^a) = (0.1, 1)$ to generate the synthetic data. Rate parameters for the network and attributes were both set to be 0.5.

We also implemented the MoM learning algorithm in Snijders et al. [2007] and compared the result to our important sampling method. Note that the learning algorithm in Snijders et al. [2007] can only deal with evenly-spaced, fully observed point evidence. Therefore, for a comparison, the observation dataset was generated by sampling a full trajectory

but only recording the values at regular intervals of Δt. We randomly sampled another 100 full trajectories as testing data.

Learning accuracy was tested by calculating the log-likelihood of the testing trajectories under the estimated models. Figure 6.1 shows the results for different amounts of data and different values of Δt. We used 400 samples and averaged across 4 runs (although the values were very stable across runs).

Our algorithm outperforms MoM almost all the time. Its accuracy is much higher than the MoM when the number of observation is small. Since samples generated by our algorithm fully agree with the observations, they provide more accurate information about the system, which is valuable when observation data are limited. When the time interval is relatively small and there are enough observation data, our algorithm can accurately estimate the model. As Δt increases, the estimation accuracy drops. The expected time between transitions for any given variable is 2 time units. Therefore, when $\Delta t = 8$, it is difficult to estimate the number of changes in the network between observations, thus explaining the poor performance of both algorithms.

Real Social Data Example

We then applied our algorithm to the "50 girls data" from the *Teenage Friends and Lifestyle Study* [Michell and Amos, 1997]. The dataset measures the changes in a network of 50 school girls, along with time-variant attributes such as smoking habits and alcohol consumption. The data contains three observations spaced one year apart. In this paper, we concentrate

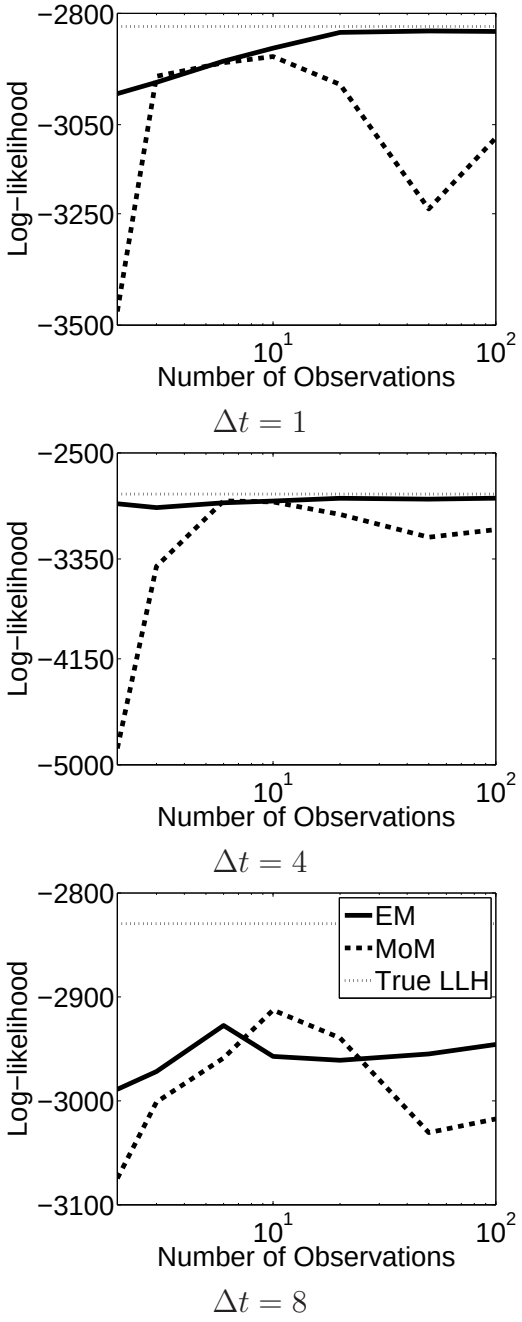


Figure 6.1: Log-likelihood of testing data as a function of the number of training data intervals.

Network Parameters			Attribute Parameters		
λ^n	Rate/Actor	0.019	λ^a	Rate/Actor	0.004
β_1^n	Density	-2.39	β_1^a	Tendency	0.14
β_2^n	Reciprocity	2.15	β_2^a	Similarity	1.17
β_3^n	Similarity	0.53			

Figure 6.2: Estimated parameters for the 50 girls dataset.

on exploring the effect between the network and the level of alcohol consumption. Alcohol consumption was measured by self-reported questions on a scale ranging from 1 (never) to 5 (more than once a week). We consider the same effects (features) as in the previous experiment. That is, the utility functions have the same format as Equation 6.6. The true parameters are obviously unknown.

Since this model describes the dynamics of all the links between any two actors in the network and the alcohol attribute for every actor, the model contains 2500 variables; 50 have 5 values and the remainder are binary. No existing exact inference algorithm can handle a system with so many variables. We ran the EM algorithm with our Metropolis-Hastings algorithm on this model using 400 samples. The estimated parameters are shown in Figure 6.2. The time unit was one day. We can see that, on average, a student reconsiders her friendships every one to two months, and a student's alcohol consumption level remains unchanged for approximately 8 months. Parameter β_2^a indicates that students tend to change their attributes so that they will be similar to their friends. The network parameters β^n show that people strongly prefer to build reciprocated connections and they are unlikely to build a connection with someone arbitrary. These parameters seem reasonable and roughly match the rates found by Snijders et al. [2007].

6.3.2 Hidden Social Network Dynamics Model

To evaluate the hidden social network dynamic model, we used two real world datasets: the Enron email dataset [Shetty and Adibi, 2004] and the reality mining dataset [Eagle and Pentland, 2006].

Enron Dataset

The Enron email dataset contains emails from 151 employees between 1998 and 2002. Email events are considered as the observation variable in the hidden social network dynamic model, which are fully observed in the Enron email dataset. Usually, this type of observation is just a set of instantaneous events. There is no "state" for this kind of variable. We use "toggle variables" to model such variables as a continuous-time Markov process. That is, each variable $O_{ij}(t)$ is a binary variable containing two indistinguishable states o_0 and o_1. The intensities with which the variable transitions in both states are required to be the same. The instantaneous event is represented as the variable flipping between states. That is, the intensity matrix for $O_{ij}(t)$ is

$$Q^{obs}_{ij|Y_{ij}=k,Y_{ji}=l} = \begin{bmatrix} -q^{obs}_{kl} & q^{obs}_{kl} \\ q^{obs}_{kl} & -q^{obs}_{kl} \end{bmatrix}$$

where $k, l \in \{0, 1\}$.

q_{kl}^{obs} in the conditional intensity matrix of O_{ij} can be estimated as

$$q_{kl}^{obs} = \frac{\sum_{i \neq j} M_{O_{ij}}[o_0|Y_{ij} = k, Y_{ji} = l] + \sum_{i \neq j} M_{O_{ij}}[o_1|Y_{ij} = k, Y_{ji} = l]}{\sum_{i \neq j} T_{O_{ij}}[o_0|Y_{ij} = k, Y_{ji} = l] + \sum_{i \neq j} T_{O_{ij}}[o_1|Y_{ij} = k, Y_{ji} = l]}.$$

We preprocessed the data as follows: We only chose emails sent in 2001 since most of the emails were sent in that year. We removed emails whose sender and recipient were the same. Emails that were sent to many recipients were usually general notices such as a reminder of a presentation at certain time and not indicative of a working relationship. If the number of recipients in an email is larger than a threshold t_r, we filtered out that email. In our experiment, we set t_r be 5. For the rest of the emails which were sent to multiple recipients, we treated them as multiple single-recipient e-mails and randomly jittered the sent times. We took the intersection of people who sent at least 100 emails in 2001 and people who received at least 100 emails in 2001 as the set of the actors in our model. Emails among these actors were used as our observation. After this process, we obtained a dataset containing 6738 emails for 31 people. We set the time unit to be one day in this experiment.

We considered four effects on the network utility function: the density effect, the reciprocity effect, the activity effect, and the popularity effect. Therefore, the utility function is

$$f^n(\beta^n, \boldsymbol{y}) = \sum_j (\beta_1^n y_{ij} + \beta_2^n y_{ij} y_{ji} + \beta_3^n y_{ij} \sum_k y_{jk} + \beta_4^n y_{ij} \sum_k y_{kj}) \ . \tag{6.7}$$

We ran Monte Carlo EM using the Metropolis-Hastings sampler with 400 samples for each iteration. We took one sample for every 1000 steps of MCMC. Before starting the EM

λ^n	Rate/Actor	0.031	$k,\ l$	q_{kl}^{obs}
β_1^n	Density	-2.362	0, 0	0.002
β_2^n	Reciprocity	1.210	0, 1	0.023
β_3^n	Activity	0.115	1, 0	0.296
β_4^n	Popularity	0.119	1, 1	0.604

Figure 6.3: Estimated parameters for the Enron dataset

iterations, we sampled 10000 samples as the "burn-in" iterations. In each EM iteration, we used the last trajectory in previous iteration as our initial sample and used 1000 samples as the "burn-in" iterations. We randomly picked the initial parameters for the model and ran the EM algorithm 5 times. All the 5 runs had very similar results. We took the average for each parameter as our learned parameters. The learned parameters are listed in Figure 6.3.

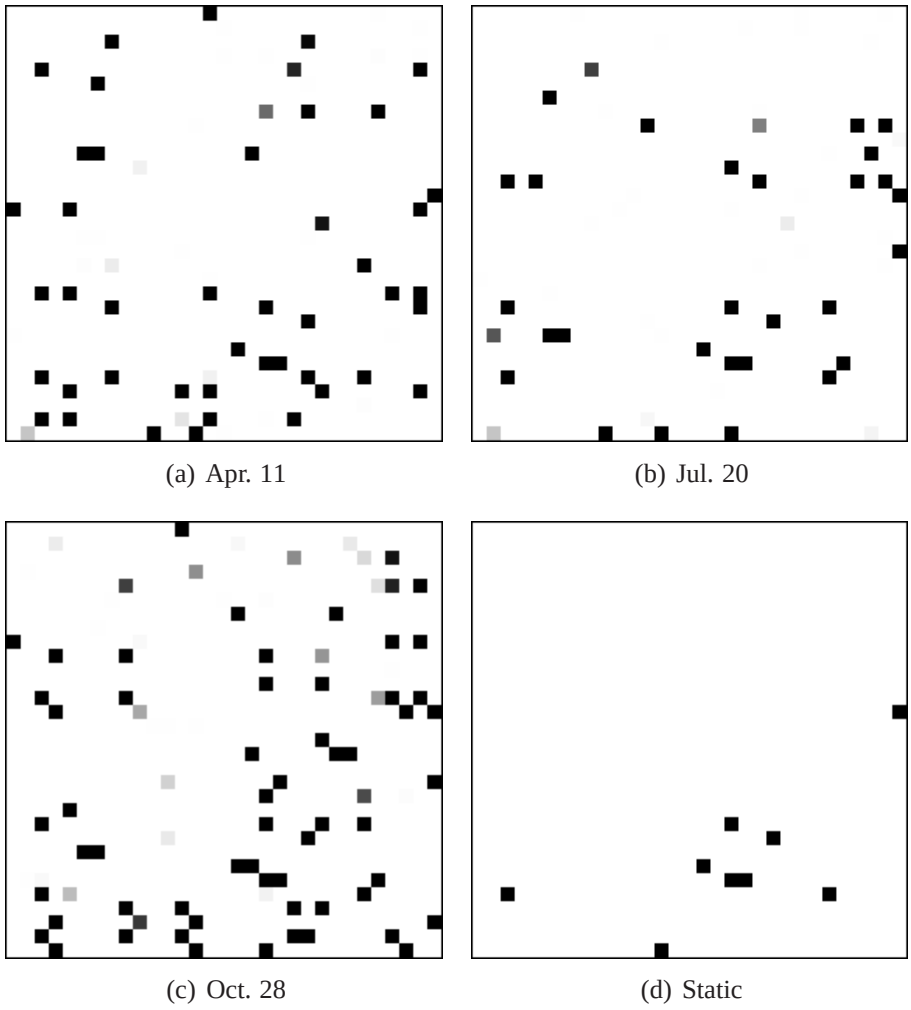
We can see that, on average, a person considers changing a working relationship about every month. When choosing the connection to change, the person is very unlikely to build a random connection ($\beta_1^n = -2.1$) and prefers to build reciprocated connections ($\beta_2^n = 1.5$). These results are very similar to what we learned from the "50 girls dataset." The popularity and activity of an actor has a positive effect (β_3^n and β_4^n are multiplied by the number of connections to or from an actor, so they can be smaller than β_2^n and still have similar impact). On average, a person A will send an email to another person B about every 3 days if there is a connection from A to B. If there is a reciprocated connection, A will send an email to B at least every 2 days. If there is no connection from A to B, it is unlikely that A will send an email to B.

We used the learned parameters as the true parameters of the model. Starting with a random trajectory, we ran our MCMC algorithm for 1,000,000 "burn in" iterations. Then

we drew a sample every 1000 iterations. We repeated this until we sampled 1000 trajectories. Using the 1000 trajectories, we calculated the probability $P(Y_{ij}(t) = 1)$, for $i, j \in \{1, \ldots, 31\}, i \neq j$. Figure 6.4 shows the network structures as matrices at three different times throughout the year. Darker spots represent higher connection probability. We consider a connection as static if the probability $P(Y_{ij}(t) = 1)$ is larger than 0.95 at all three time points. Figure 6.4(d) shows all the static links among the actors. Since the Enron dataset represents working groups changing over one year, we can see that the three matrices are different, showing that the links among people are dynamic processes, but there are some stable connections.

Reality Mining Dataset

The reality mining dataset contains mobile phone usage information for 97 people from June 2004 to May 2005. The information includes call logs, bluetooth devices in proximity, cell phone locations, application usage, and more. The dataset was collected by installing software in the participants' cellphones. The participants were all from MIT. A majority of them are from the MIT Media Laboratory. The remaining are incoming students at the MIT Sloan Business School. In our experiment, we only used the call logs as our observations. We modeled the dynamics of the observation variable (phone calls of each participant) as described in Example 6.2.1. We use the same utility function as in Equation 6.7. Since each participant could start and stop using the data collection software at anytime, the call log for each participant was only partially observed.



(a) Apr. 11 (b) Jul. 20

(c) Oct. 28 (d) Static

Figure 6.4: Enron adjacency matrix at different times.

λ^n	Rate/Actor	0.0010
β_1^n	Density	-3.3254
β_2^n	Reciprocity	2.7963
β_3^n	Activity	0.0881
β_4^n	Popularity	0.1382

Figure 6.5: Parameters of utility function for reality mining dataset

We preprocessed the dataset as the following. Since call logs can only be identified using phone number identifiers, we only chose those who have a valid phone number identifier from the 97 participants. For phone numbers whose owners are not listed as participants, we chose those who were contacted by at least three of the selected participants, which gave us 13 actors whose identification is unknown. In total, there are 92 actors in our model, 25 of which are from the MIT Sloan Business School, 54 from the MIT Media Lab, and 13 unknown actors. We selected all the call logs among these 92 actors and remove duplications (a phone-call may be recorded twice, on both the caller's and callee's phones). The observation data contains 6488 events. We set the time unit to be one hour in this experiment.

We started the Metropolis-Hastings sampler with a randomly generated trajectory and discarded the first 50000 samples as "burn in" samples. Then, in each EM iteration, we used 50000 samples to estimate the parameters of the rate function. We sampled 50000 samples, keeping only one out of every ten samples, to estimate the conditional intensity matrix of the observation variable. For the parameters of the utility function, we estimated them by sampling 50000 trajectories and took one for every 200 samples. During each EM iteration, we sampled another 20000 samples as "burn in" iterations. We ran the MCEM algorithms 5 times. Each run was started with random parameters.

The estimated parameters of utility function are shown in Figure 6.5. On average, each actor would change a relationship about every 40 days. The estimated parameters of the utility function are similar to those in Enron dataset. However, in this model, people are less likely to create a random tie ($\beta_1^n = -3.3254$) and are more likely to form reciprocated connections ($\beta_2^n = 2.7963$). People prefer the "popularity effect" ($\beta_4^n = 0.1382$) more than the "activity effect" ($\beta_3^n = 0.0881$) in this model.

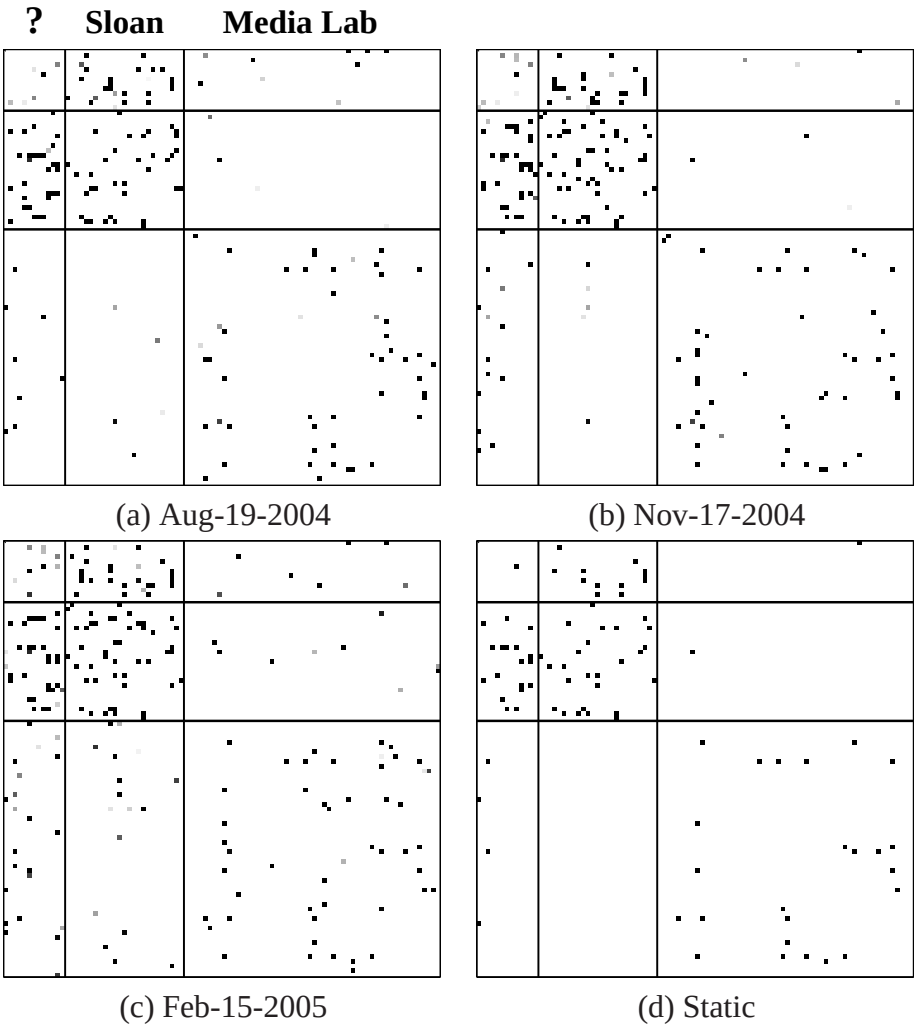
The conditional intensity matrices of the observation variable are shown in Figure 6.6. If there is a reciprocated link between actor A and B, A will try to contact B about every 3 or 4 days. About 20 percent of the time, A will send a text message, and A will call B directly the remaining 80 percent of the time. On average, each conversation lasts for 5 minutes. If there is only a single link $A \rightarrow B$ between A and B, A contacts B about every 10 days on average. If there is no connection between A and B, they almost never call each other.

Similar to the experiment on the Enron dataset, we used the estimated parameters as the true parameters of the model and calculated the probability $P(Y_{ij}(t) = 1)$, for $i, j = 1, \ldots, 92, i \neq j$ using the Metropolis-Hastings sampling algorithm. We sampled 10000 trajectories to estimate the probability. Adjacency matrices of the network at different time are shown in Figure 6.7. Figure 6.7(d) shows the static connections in this model.

In the matrices, actors are aligned according to their identifications. The first 13 rows (and columns) represent actors whose identification are unknown. The next 25 rows represent students from the MIT Sloan Business School and the last 54 rows represent people from the MIT Media Lab. The three matrices are different. We can see that there are some stable links

$k\,,l$	Q_{kl}^{obs}
$0\,,0$	$\begin{bmatrix} -0.3094\times10^{-6} & 0.0074\times10^{-6} & 0.2998\times10^{-6} & 0.0022\times10^{-7} \\ 977.5440 & -977.5440 & 0 & 0 \\ \infty & 0 & -\infty & 0 \\ \infty & 0 & 0 & -\infty \end{bmatrix}$
$0\,,1$	$\begin{bmatrix} -0.1723\times10^{-3} & 0.1232\times10^{-3} & 0.0144\times10^{-3} & 0.0347\times10^{-3} \\ 977.8032 & -977.8032 & 0 & 0 \\ \infty & 0 & -\infty & 0 \\ \infty & 0 & 0 & -\infty \end{bmatrix}$
$1\,,0$	$\begin{bmatrix} -0.0960 & 0.0792 & 0.0144 & 0.0024 \\ 999.5496 & -999.5496 & 0 & 0 \\ \infty & 0 & -\infty & 0 \\ \infty & 0 & 0 & -\infty \end{bmatrix}$
$1\,,1$	$\begin{bmatrix} -0.3000 & 0.2088 & 0.0624 & 0.0288 \\ 287.3160 & -287.3160 & 0 & 0 \\ \infty & 0 & -\infty & 0 \\ \infty & 0 & 0 & -\infty \end{bmatrix}$

Figure 6.6: Conditional intensity matrix of reality mining dataset. (Time unit is one day)



(a) Aug-19-2004 (b) Nov-17-2004

(c) Feb-15-2005 (d) Static

Figure 6.7: Reality mining adjacency matrix at different times.

in the network. However, we also observe many relationship change with time, which shows that the network structure is dynamic. Most of the links reside in the diagonal boxes in the figure. This is very reasonable since people usually only have ties with people within their community. This to some extent verifies the correctness of our model.

6.4 Conclusion

We provide a sampling-based learning algorithm for continuous-time social network models and provide results for a model with 2500 variables. We also develop a hidden social network dynamics model in which indirect observation of the network can be used, and we develop an MCMC sampling algorithm for it. Our method is simple and easy to implement. The idea of the algorithm is general and can be easily extended to other continuous-time systems. For social networks, we provide a learning method that is better than the previous method of moments estimation, particularly when data is scarce (a common occurrence in sociology studies). We demonstrate results on learning a dynamic social network with approximately 9000 variables from indirect observations. These two advances greatly extend the types of social dynamic data that can be analyzed.

Chapter 7

Conclusion

Continuous time Bayesian networks (CTBNs) allow us to model dynamic systems in continuous time. Their structured representation also allows us to exploit the independencies among the variables of a dynamic system. However, exact inference in CTBNs is often intractable for large systems. Therefore, an approximate inference algorithm is needed in many applications.

In this dissertation, we have presented an approximate inference algorithm for CTBNs using importance sampling. Our algorithm generates weighted samples by naturally simulating the CTBN we are reasoning about. The approximate expectation of any function of the trajectory can be calculated using these weighted samples. We naturally extended the algorithm to sequential Monte Carlo methods, such as particle filtering and smoothing. A Metropolis-Hastings algorithm for CTBNs was also developed based on the importance sampling algorithm.

We then applied CTBN models and our approximate inference algorithm to learn social network dynamics. We argued that a number of sociology models for social network dynamics can be viewed as time-variant CTBNs. Therefore, our importance sampling algorithm can be used to develop a maximum likelihood estimation algorithm. We also proposed the hidden social network dynamic model, which allows indirect observations of social networks. Our experiments on real world data, such as emails and phone-call logs among people, showed that our model learned social network dynamics effectively.

As we can see from Chapter 4, the networks used in our experiments are already at the limit size for exact inference method. Real world applications, such as the social networks we dealt with in Chapter 6, often contain thousands of variables. Thus, approximate inference approaches are very important for reasoning about dynamic systems. The algorithms that we developed using importance sampling provide an simple way to reason about complex dynamic systems. Our algorithms differ from other approximate inference approaches for CTBNS in a number of key ways. In our algorithms, samples are generated by naturally simulating the dynamic system we are reasoning about. Thus, our algorithm does not depend on complex numeric computations. The transition times for variables are sampled from regular exponential distributions in our algorithm, which can be done in constant time. Our importance sampling based algorithms are fast and simple to implement. They can be used to calculate the expected value of any function of a trajectory, including the expected sufficient statistics that are necessary for expectation-maximization for parameter estimation with missing data. The idea of our importance sampling algorithm is general; it can be ex-

tended to other continuous-time dynamic systems. Besides social network dynamic models, as demonstrated in the present dissertation, the idea of our algorithm has also been applied to inference in a continuous-time probabilistic programming language [Pfeffer, 2009].

For social networks, our learning method not only achieves better estimation accuracy than previous methods but also can deal with data with arbitrary observations (including duration observations and missing observations), which is impossible for previous social network estimation methods. In addition, our hidden social network dynamic model utilizes social events among people as indirect observations to learn social network dynamics. Since missing observations and asynchronous observations are very common in social network data, our model allows more types of social dynamic data to be analyzed.

The proposal distribution we used in our importance sampling algorithm is based on forward sampling algorithm. We only "force" the behavior of the variable we are sampling according to its upcoming evidence. The advantage of this proposal distribution is that it is very fast and easy to implement. However, only considering the upcoming evidence of the sampled variable sometimes is misleading. This is especially true when the correlation among variables are strong. We can obtain a better proposal distribution by considering the evidence of both the variable itself and those in its Markov blanket. However, this involves more variables which requires more complex computations, which results in longer running times. It would be very intriguing to find a method to balance the computation complexity and the goodness of the proposal distribution.

Bibliography

Carolyn J. Anderson, Stanley Wasserman, and Bradley Crouch. A p* primer: logit models for social networks. *Social Networks*, 21:37–66, January 1999.

K.O. Bowman and L.R. Shenton. Method of moments. *Encyclopedia of Statistical Sciences*, 5:467–473, 1985.

Ido Cohn, Tal El-Hay, Nir Friedman, and Raz Kupferman. Mean field variational approximation for continuous-time Bayesian networks. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, 2009.

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

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

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

Nathan Eagle and Alex Sandy Pentland. Reality mining: sensing complex social systems. *Personal Ubiquitous Computing*, 10(4):255–268, 2006.

Tal El-Hay, Nir Friedman, and Raz Kupferman. Gibbs sampling in factorized continuous-time Markov processes. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, pages 169–178, 2008.

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

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

Yu Fan and Christian R. Shelton. Learning continuous-time social network dynamics. In *Proceedings of the Twenty-Fifth International Conference on Uncertainty in Artificial Intelligence*, 2009.

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

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

Fan Guo, Steve Hanneke, Wenjie Fu, and Eric Xing. Recovering temporally rewiring networks: A model-based approach. In *Proceedings of the 24th International Conference on Machine Learning*, pages 321–328, 2007.

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

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

Peter D. Hoff, Adrian E. Raftery, and Mark S. Handcock. Latent space approaches to social network analysis. *Journal of the American Statistical Association*, 97:1090–1098, December 2002.

Paul W. Holland and Samuel Leinhardt. An exponential family of probability distributions for directed graphs. *Journal of the American Statistical Association*, 76(373):33–50, March 1981.

Mark Huisman and Christian Steglich. Treatment of non-response in longitudinal network studies. *Social Networks*, 20:297–308, October 2008.

David R. Hunter and Mark S. Handcock. Inference in curved exponential family models for networks. *Journal of Computational Graphical Statistics*, 15:565–583, September 2006.

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

Johan H. Koskinen and Tom A.B. Snijders. Bayesian inference for dynamic social network data. *Journal of Statistical Planning and Inference*, 137:3930–3938, 2007.

David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management*, pages 556–559, 2003.

Lynn Michell and Amanda Amos. Teenage friends and lifestyle study dataset. http://stat.gamma.rug.nl/snijders/s50data.htm, 1997.

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

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

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

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

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

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

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

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

Judea Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kauffman, 1988.

Avi Pfeffer. Ctppl: A continuous time probabilistic programming language. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 1943–1950, 2009.

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C*. Cambridge University Press, second edition, 1992.

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

Purnamrita Sarkar and Andrew W. Moore. Dynamic social network analysis using latent space models. *SIGKDD Explor. Newsl.*, 7(2):31–40, 2005.

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

Jitesh Shetty and Jafar Adibi. The Enron email dataset database schema and brief statistical report, 2004. http://www.isi.edu/~adibi/Enron/Enron.htm.

Tom A.B. Snijders. *Models for Longitudinal Network Data*, chapter 11. Cambridge Univ. Press, New York, 2005.

Tom A.B. Snijders, Christian E.G. Steglich, and Michael Schweinberger. Modeling the co-evolution of networks and behavior. In *Longitudinal models in the behavioral and related sciences*, chapter 4. Lawrence Erlbaum, 2007.

Christian Steglich, Tom A. B. Snijders, and Patrich West. Applying SIENA: An illustrative analysis of the co-evolution of adolescents' friendship networks, taste in music, and alcohol consumption. *Methodology*, 2:48–56, 2006.

Stanley Wasserman. A stochastic model for directed graphs with transition rates determined by reciprocity. In K. Schuessler, editor, *Sociological Methodology*, pages 392–412. Jossy-Bass, 1979.

Stanley Wasserman. Analyzing social networks as stochastic processes. *Journal of the American Statistical Association*, 75:280–294, 1980.

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

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

Appendix A

Background Materials

A.1 Markov Chain Monte Carlo

Let $\boldsymbol{X}$ be a vector of random variables with state space $Val(\boldsymbol{X})$. Let π be a distribution over $\boldsymbol{X}$. To estimate the expectation of a function $f(\boldsymbol{X})$

$$E[f(\boldsymbol{X})] = \int f(\boldsymbol{x})\pi(\boldsymbol{x})d\boldsymbol{x}\,, \tag{A.1}$$

the Markov chain Monte Carlo method generates a sequence of samples $\boldsymbol{x}^{(t)} \in Val(\boldsymbol{X})$, $t = 1, \ldots, n$ by constructing a Markov chain and running it for a long time. The Markov chain is constructed such that its stationary distribution is the desired distribution $\pi(\boldsymbol{X})$.

Then Equation A.1 can be approximated by

$$E[f(\boldsymbol{X})] \approx \frac{1}{n}\sum_{t=1}^{n} f(\boldsymbol{x}^{(t)})\,.$$

A.1.1 Markov Chain

Let $\{\boldsymbol{X}^{(0)}, \boldsymbol{X}^{(1)}, \boldsymbol{X}^{(2)}, \ldots\}$ be a sequence of random variables such that at each time $t \geq 0$, the distribution of the next state $\boldsymbol{X}^{(t+1)}$ depends only on the current state $\boldsymbol{X}^{(t)}$. That is, given the current state $\boldsymbol{X}^{(t)}$, the next state $\boldsymbol{X}^{(t+1)}$ is independent of the past states $\{\boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(t-1)}\}$. We denote this distribution as $\mathcal{T}(\boldsymbol{X}^{(t)} \rightarrow \boldsymbol{X}^{(t+1)})$. The sequence is called a Markov chain. The distribution $\mathcal{T}$ is called the transition model of the chain. We assume that $\mathcal{T}$ is time-invariant.

Assume that $\boldsymbol{X}^{(0)}$, the state at $t = 0$, is from some initial distribution $P^{(0)}(\boldsymbol{X}^{(0)})$. The distribution of the state at each step t is

$$P^{(t+1)}(\boldsymbol{X}^{(t+1)} = \boldsymbol{x}') = \int P^{(t)}(\boldsymbol{X}^{(t)} = \boldsymbol{x})\mathcal{T}(\boldsymbol{x} \rightarrow \boldsymbol{x}')d\boldsymbol{x}\,.$$

A finite-state Markov chain is called an ergodic chain if there exists t such that for any state $\boldsymbol{x}$ and $\boldsymbol{x}'$, $P^{(t)}(\boldsymbol{X}^{(t)} = \boldsymbol{x}') > 0$. If we let an ergodic Markov chain run for a long time, it converges to a unique stationary distribution which does not depend on $\boldsymbol{X}^{(0)}$. In particular,

let $\phi(\boldsymbol{X})$ be the stationary distribution of the Markov chain. $\phi(\boldsymbol{X})$ satisfies

$$\phi(\boldsymbol{X} = \boldsymbol{x}') = \int \phi(\boldsymbol{X} = \boldsymbol{x})\mathcal{T}(\boldsymbol{x} \rightarrow \boldsymbol{x}')d\boldsymbol{x}\,. \tag{A.2}$$

If a Markov chain $\mathcal{T}$ and a distribution π satisfy the following

$$\pi(\boldsymbol{x})\mathcal{T}(\boldsymbol{x} \rightarrow \boldsymbol{x}') = \pi(\boldsymbol{x}')\mathcal{T}(\boldsymbol{x}' \rightarrow \boldsymbol{x})\,, \tag{A.3}$$

then π and $\mathcal{T}$ satisfy Equation A.2, since

$$\begin{aligned}\int_{\boldsymbol{x}\in Val(\boldsymbol{X})} \pi(\boldsymbol{x})\mathcal{T}(\boldsymbol{x} \rightarrow \boldsymbol{x}')d\boldsymbol{x} &= \int_{\boldsymbol{x}\in Val(\boldsymbol{X})} \pi(\boldsymbol{x}')\mathcal{T}(\boldsymbol{x}' \rightarrow \boldsymbol{x})d\boldsymbol{x} \\ &= \pi(\boldsymbol{x}') \int_{\boldsymbol{x}\in Val(\boldsymbol{X})} \mathcal{T}(\boldsymbol{x}' \rightarrow \boldsymbol{x})d\boldsymbol{x} \\ &= \pi(\boldsymbol{x}')\,.\end{aligned}$$

Equation A.3 is called the detailed balance equation.

A.1.2 MCMC Sampler

Equation A.3 shows us one way to generate a sequence of dependent samples from a distribution π. The samples are generated through a Markov chain having π as its stationary distribution. This is called Markov chain Monte Carlo (MCMC).

Specifically, suppose we would like to generate samples from a target distribution $\pi(\boldsymbol{X})$. We define a Markov chain via the state space of $Val(\boldsymbol{X})$. For each pair of states $\boldsymbol{x}, \boldsymbol{x}' \in Val(\boldsymbol{X})$, we construct the transition model $\mathcal{T}$ of the Markov chain such that Equation A.3 is satisfied. If we run the Markov chain for a long time, the distribution of the state sequence approaches to the desired distribution π.

Given the initial state distribution $P^{(0)}(\boldsymbol{X})$ and the transition model $\mathcal{T}$, the MCMC sampling procedure can be described as follows.

1. Sample $\boldsymbol{x}^{(0)}$ from $P^{(0)}(\boldsymbol{X})$.

2. Repeat the following N times:

   Sample $\boldsymbol{x}^{(t+1)}$ from $\mathcal{T}(\boldsymbol{x}^{(t)} \rightarrow \boldsymbol{X})$.

3. Return $\boldsymbol{x}^{(0)}, \ldots, \boldsymbol{x}^{(N)}$.

Typically, the initial samples are not completely valid because the Markov Chain has not converged. Samples at the beginning iterations of a MCMC run are usually thrown away. These initial samples are called "burn-in" samples.

Different choices of the transition model $\mathcal{T}$ of the Markov chain result in different MCMC algorithms. We introduce the Gibbs sampling algorithm and the Metropolis-Hastings algorithm in the following two sections.

A.1.3 Gibbs Sampling Algorithm

Suppose we can decompose $\boldsymbol{X}$ into d components $(X_1, \ldots, X_d)$. The Gibbs sampler defines the transition probability of the Markov chain to be the conditional distribution $\pi(X_i|\boldsymbol{X}_{-i})$, where $\boldsymbol{X}_{-i}$ represents all the variables in $(X_1, \ldots, X_d)$ except X_i. That is, in each iteration, the Gibbs sampler only changes the value of one component and fixes the values of other variables as evidence.

Formally, we describe the Gibbs sampler as follows. Let $\boldsymbol{x}^{(t)} = (\boldsymbol{x}_1^{(t)}, \ldots, \boldsymbol{x}_d^{(t)})$ be the value of $\boldsymbol{X}$ at iteration t. Let $\boldsymbol{x}_{-i}^{(t)}$ be the values of all the components at t except X_i. At iteration $t+1$, we do the following steps:

- Randomly pick i from $(1, \ldots, d)$ with probability $1/d$.
- Sample $x_i^{(t+1)}$ from $\pi(X_i|\boldsymbol{X}_{-i} = \boldsymbol{x}_{-i}^{(t)})$, while leaving all the other variables unchanged. That is, let $\boldsymbol{x}_{-i}^{(t+1)} = \boldsymbol{x}_{-i}^{(t)}$.

It can be shown that at every step, the detailed balance condition is satisfied.

$$\begin{aligned}\pi(x_i^{(t)}, \boldsymbol{x}_{-i}^{(t)})\pi(x_i^{(t+1)}|\boldsymbol{x}_{-i}^{(t)}) =& \pi(\boldsymbol{x}_{-i}^{(t)})\pi(x_i^{(t)}|\boldsymbol{x}_{-i}^{(t)})\pi(x_i^{(t+1)}|\boldsymbol{x}_{-i}^{(t)}) \\ =& \pi(x_i^{(t+1)}, \boldsymbol{x}_{-i}^{(t)})\pi(x_i^{(t)}|\boldsymbol{x}_{-i}^{(t)})\end{aligned}$$

A.1.4 Metropolis-Hastings Sampling Algorithm

The Gibbs sampling algorithm requires sampling from the conditional distribution $\pi(X_i|\boldsymbol{X}_{-i})$. However, in many applications, samples cannot be generated from $\pi(X_i|\boldsymbol{X}_{-i})$ efficiently.

The Metropolis-Hastings algorithm provides a more general way to construct the Markov chain with the desired stationary distribution.

Unlike the Gibbs sampler, the Metropolis-Hastings algorithm generates the next-state samples using a proposal distribution $\mathcal{T}^Q$, where $\mathcal{T}^Q$ defines a distribution over possible successor states in $Val(\boldsymbol{X})$. The Metropolis-Hastings sampler starts with a random state. Let $\boldsymbol{x}^{(t)}$ be the state at iteration t. In each iteration, the Metropolis-Hastings algorithm performs the following two steps:

- Sample $\boldsymbol{x}'$ from the proposal distribution $\mathcal{T}^Q(\boldsymbol{x}^{(t)} \to \boldsymbol{x}')$.

- Draw u from the uniform distribution $\mathbf{u}(0, 1)$ and update

$$\boldsymbol{x}^{(t+1)} = \begin{cases} \boldsymbol{x}' & \text{if } u \leq \mathcal{A}(\boldsymbol{x}^{(t)} \to \boldsymbol{x}') \\ \boldsymbol{x}^{(t)} & \text{otherwise.} \end{cases}$$

  where the acceptance probability $\mathcal{A}(\boldsymbol{x} \to \boldsymbol{x}')$ is defined as

$$\mathcal{A}(\boldsymbol{x} \to \boldsymbol{x}') = \min\left[1, \frac{\pi(\boldsymbol{x}')\mathcal{T}^Q(\boldsymbol{x}' \to \boldsymbol{x})}{\pi(\boldsymbol{x})\mathcal{T}^Q(\boldsymbol{x} \to \boldsymbol{x}')}\right].$$

For any state $\boldsymbol{x} \neq \boldsymbol{x}'$, the transition model $\mathcal{T}$ for the Metropolis-Hastings algorithm is

$$\mathcal{T}(\boldsymbol{x} \to \boldsymbol{x}') = \mathcal{T}^Q(\boldsymbol{x} \to \boldsymbol{x}')\min\left[1, \frac{\pi(\boldsymbol{x}')\mathcal{T}^Q(\boldsymbol{x}' \to \boldsymbol{x})}{\pi(\boldsymbol{x})\mathcal{T}^Q(\boldsymbol{x} \to \boldsymbol{x}')}\right].$$

Hence,

$$\begin{aligned}\pi(\boldsymbol{x})\mathcal{T}(\boldsymbol{x}\rightarrow\boldsymbol{x}') =& \pi(\boldsymbol{x})\mathcal{T}^{Q}(\boldsymbol{x}\rightarrow\boldsymbol{x}')\text{min}\left[1,\frac{\pi(\boldsymbol{x}')\mathcal{T}^{Q}(\boldsymbol{x}'\rightarrow\boldsymbol{x})}{\pi(\boldsymbol{x})\mathcal{T}^{Q}(\boldsymbol{x}\rightarrow\boldsymbol{x}')}\right]\\ =& \text{min}\left[\pi(\boldsymbol{x}')\mathcal{T}^{Q}(\boldsymbol{x}'\rightarrow\boldsymbol{x}),\pi(\boldsymbol{x})\mathcal{T}^{Q}(\boldsymbol{x}\rightarrow\boldsymbol{x}')\right],\end{aligned}$$

which is a symmetric function in $\boldsymbol{x}$ and $\boldsymbol{x}'$. Thus, the detailed balance condition is satisfied.

If $\boldsymbol{X}$ can be decomposed into d components $(X_1,\ldots,X_d)$, it is often more convenient and computationally efficient to update one component a time. Let $x_i^{(t)}$ denote the value of X_i at iteration t. Let $\boldsymbol{x}_{-i}^{(t)}$ be the values of the remaining variables. In each iteration, we pick a component X_i, fix the value of all the other components, and update X_i according to its proposal distribution $Q_i(X_i|x_i^{(t)},\boldsymbol{x}_{-i}^{(t)})$. We accept the new sampled state x_i' of X_i with probability

$$\mathcal{A}((x_i^{(t)},\boldsymbol{x}_{-i}^{(t)})\rightarrow(x_i',\boldsymbol{x}_{-i}^{(t)})) = \text{min}\left[1,\frac{\pi((x_i',\boldsymbol{x}_{-i}^{(t)}))Q_i((x_i',\boldsymbol{x}_{-i}^{(t)})\rightarrow(x_i^{(t)},\boldsymbol{x}_{-i}^{(t)}))}{\pi((x_i^{(t)},\boldsymbol{x}_{-i}^{(t)}))Q_i((x_i^{(t)},\boldsymbol{x}_{-i}^{(t)})\rightarrow(x_i',\boldsymbol{x}_{-i}^{(t)}))}\right].$$

Notice that if we pick the proposal distribution Q_i to be $\pi(X_i|\boldsymbol{x}_{-i}^{(t)})$, we accept the new state of X_i with probability 1 in each iteration, which is just the Gibbs sampling algorithm. Thus, the Gibbs sampling algorithm is a special case of the Metropolis-Hastings algorithm.

A.2 Expectation Maximization

The expectation maximization (EM) algorithm [Dempster et al., 1977] is an iterative optimization method to estimate the parameters of probabilistic models in the presence of missing or hidden data. Let θ be the parameter of the probabilistic model we are trying to estimate. Let $\mathcal{X}$ denote the observed incomplete data. Let $\mathcal{Y}$ denote an assignment to the unobserved variables. The EM algorithm finds the parameter θ^* such that the posterior probability of θ given the data $\mathcal{X}$ is maximized. That is,

$$\theta^* = \arg\max_{\theta} \sum_{\mathcal{Y}} P(\theta, \mathcal{Y}|\mathcal{X}) \,. \tag{A.4}$$

Notice that maximizing Equation A.4 is equivalent to maximizing the logarithm of the joint distribution

$$\theta^* = \arg\max_{\theta} \log P(\theta, \mathcal{X}) = \arg\max_{\theta} \log \sum_{\mathcal{Y}} P(\theta, \mathcal{Y}, \mathcal{X}) \tag{A.5}$$

since $P(\theta, \mathcal{X}) \propto P(\theta|\mathcal{X})$ and the logarithm function is monotonically increasing.

We can rewrite the logarithm of the joint distribution as

$$\log P(\theta, \mathcal{X}) = \log \sum_{\mathcal{Y}} P(\theta, \mathcal{Y}, \mathcal{X}) = \log \sum_{\mathcal{Y}} q^t(\mathcal{Y}) \frac{P(\theta, \mathcal{Y}, \mathcal{X})}{q^t(\mathcal{Y})} \,,$$

where $q^t(\mathcal{Y})$ is an arbitrary probability distribution over $\mathcal{Y}$. Using Jensens inequality[1], we have

$$\begin{aligned} \log P(\theta, \mathcal{X}) &= \log \sum_{\mathcal{Y}} q^t(\mathcal{Y}) \frac{P(\theta, \mathcal{Y}, \mathcal{X})}{q^t(\mathcal{Y})} \\ &\geq \sum_{\mathcal{Y}} q^t(\mathcal{Y}) \log \frac{P(\theta, \mathcal{Y}, \mathcal{X})}{q^t(\mathcal{Y})} \end{aligned} \tag{A.6}$$

We can see that Equation A.6 provides a lower bound for $\log P(\theta, \mathcal{X})$. The idea behind the EM algorithm is to start with some initial parameter θ^0, and alternatingly compute the lower bound in Equation A.6 and compute θ^{t+1} that maximizes the bound. The EM algorithm will eventually converge to a local maximum θ^* of the objective function, provided that the lower bound improves at each iteration.

Maximizing this lower bound with respect to q^t gives

$$q^t(\mathcal{Y}) = P(\mathcal{Y}|\theta^t, \mathcal{X}) \,. \tag{A.7}$$

This is called the expectation step (or E-step), which makes the bound tight. Having the optimal lower bound, we then maximize it with respect to θ. Replacing $q^t(\mathcal{Y})$ with $P(\mathcal{Y}|\theta^t, \mathcal{X})$ in Equation A.7, we have

[1]Jensens inequality states that, for any concave function f, $\sum_i p_i f(x_i) \geq f(\sum_i p_i x_i)$ where $\sum_i p_i = 1$.

$$\sum_{\mathcal{Y}} q^t(\mathcal{Y}) \log \frac{P(\theta, \mathcal{Y}, \mathcal{X})}{q^t(\mathcal{Y})} \tag{A.8}$$

$$= \sum_{\mathcal{Y}} P(\mathcal{Y}|\theta^t, \mathcal{X}) \log P(\theta, \mathcal{Y}, \mathcal{X}) - \sum_{\mathcal{Y}} P(\mathcal{Y}|\theta^t, \mathcal{X}) \log P(\mathcal{Y}|\theta^t, \mathcal{X})$$

$$= \sum_{\mathcal{Y}} P(\mathcal{Y}|\theta^t, \mathcal{X}) \log P(\mathcal{Y}, \mathcal{X}|\theta) P(\theta) - \sum_{\mathcal{Y}} P(\mathcal{Y}|\theta^t, \mathcal{X}) \log P(\mathcal{Y}|\theta^t, \mathcal{X})$$

$$= \sum_{\mathcal{Y}} P(\mathcal{Y}|\theta^t, \mathcal{X}) \log P(\mathcal{Y}, \mathcal{X}|\theta) + \sum_{\mathcal{Y}} P(\mathcal{Y}|\theta^t, \mathcal{X}) \log P(\theta)$$

$$- \sum_{\mathcal{Y}} P(\mathcal{Y}|\theta^t, \mathcal{X}) \log P(\mathcal{Y}|\theta^t, \mathcal{X})$$

$$= E_{P(\mathcal{Y}|\theta^t, \mathcal{X})}[\log P(\mathcal{Y}, \mathcal{X}|\theta)] + \log P(\theta) - E_{P(\mathcal{Y}|\theta^t, \mathcal{X})}[P(\mathcal{Y}|\theta^t, \mathcal{X})]$$

The first term in Equation A.8 is the expected log-likelihood of complete data with respect to $P(\mathcal{Y}|\theta^t, \mathcal{X})$. The second term is the logarithm of the prior distribution of parameter θ. The last term is the entropy of $P(\mathcal{Y}|\theta^t, \mathcal{X})$. Since the last term does not depend on θ, maximizing Equation A.8 with respect θ is equivalent to maximizing only the first two terms. Thus, our maximization problem can be written as

$$\theta^{t+1} = \arg\max_{\theta} E_{P(\mathcal{Y}|\theta^t, \mathcal{X})}[\log P(\mathcal{Y}, \mathcal{X}|\theta)] + \log P(\theta) \tag{A.9}$$

$$= \arg\max_{\theta} Q(\theta|\theta^t) + \log P(\theta)$$

where $Q(\theta|\theta^t) = E_{P(\mathcal{Y}|\theta^t, \mathcal{X})}[\log P(\mathcal{Y}, \mathcal{X}|\theta)]$. This is called the maximization step (or M-step).

The EM procedure can be summarized as follows:

1. Initialize $\theta^{(0)}$ randomly or based on some prior knowledge.

2. Iteratively improve the estimate of θ by alternating the following two steps until convergence:

   E-Step: Calculate $E_{P(\mathcal{Y}|\theta^t,\mathcal{X})}[\log P(\mathcal{Y},\mathcal{X}|\theta)]$.

   M-Step: $\theta^{t+1} = \arg\max_\theta Q(\theta|\theta^t) + \log P(\theta)$

In the E-step, we need to calculate the log-likelihood of the expected complete data. In many situations, the likelihood of the data can be calculated using sufficient statistics. For example, in the exponential family of models such as continuous time Bayesian networks, the likelihood of the data is represented using sufficient statistics as shown in Equation 3.1 and 3.2. Therefore, it is equivalent to calculating the expected sufficient statistics in the E-step.

In the M-step, the optimization incorporates the prior distribution of the parameter θ. Equation A.9 can be rewritten as

$$\begin{aligned}
\theta^{t+1} &= \arg\max_\theta E_{P(\mathcal{Y}|\theta^t,\mathcal{X})}[\log P(\mathcal{Y},\mathcal{X}|\theta)] + \log P(\theta) \\
&= \arg\max_\theta E_{P(\mathcal{Y}|\theta^t,\mathcal{X})}[\log P(\mathcal{Y},\mathcal{X}|\theta)P(\theta)] \\
&= \arg\max_\theta E_{P(\mathcal{Y}|\theta^t,\mathcal{X})}[\log \frac{P(\theta|\mathcal{Y},\mathcal{X})}{P(\mathcal{Y},\mathcal{X})}] \\
&= \arg\max_\theta E_{P(\mathcal{Y}|\theta^t,\mathcal{X})}[\log P(\theta|\mathcal{Y},\mathcal{X})]\,,
\end{aligned}$$

which means that it is a maximum a posteriori (MAP) estimation. If we assume that every θ is equally probable, the M-step is equivalent to

$$\theta^{t+1} = \arg\max_{\theta} E_{P(\mathcal{Y}|\theta^t,\mathcal{X})}[\log P(\mathcal{Y},\mathcal{X}|\theta)]\,,$$

which is a maximum likelihood estimation.