Factored Performance Functions with Structural Representation in Continuous Time Bayesian Networks

Liessman Sturlaugson and **John W. Sheppard**
Computer Science Department
Montana State University
Bozeman, MT 59717
{*liessman.sturlaugson, john.sheppard*}*@cs.montana.edu*

Abstract

The continuous time Bayesian network (CTBN) is a probabilistic graphical model that enables reasoning about complex, interdependent, and continuous-time subsystems. The model uses nodes to denote subsystems and arcs to denote conditional dependence. This dependence manifests in how the dynamics of a subsystem change based on the current states of its parents in the network. While the original CTBN definition allows users to specify the dynamics of how the system evolves, users might also want to place value expressions over the dynamics of the model in the form of performance functions. We formalize these performance functions for the CTBN and show how they can be factored in the same way as the network, allowing what we argue is a more intuitive and explicit representation. For cases in which a performance function must involve multiple nodes, we show how to augment the structure of the CTBN to account for the performance interaction while maintaining the factorization of a single performance function for each node.

Introduction

Many problems in artificial intelligence require reasoning about complex systems. One important and difficult type of system is one that changes through time. Temporal modeling and reasoning present additional challenges in representing the system's dynamics while efficiently and accurately inferring the system's behavior through time. Continuous time Bayesian networks (CTBNs) were introduced by Nodelman, Shelton, and Koller (2002) as a temporal model capable of representing and reasoning about finite- and discrete-state systems without committing to a uniform discretization of time, as found with dynamic Bayesian networks (Murphy 2002).

Since then, the CTBN has found use in a wide variety of applications. For example, they have been used for inferring users' presence, activity, and availability over time (Nodelman and Horvitz 2003); robot monitoring (Ng, Pfeffer, and Dearden 2005); modeling server farm failures (Herbrich, Graepel, and Murphy 2007); modeling social network dynamics (Fan and Shelton 2009); modeling sensor networks (Shi, Tang, and You 2010); building intrusion detection systems (Xu and Shelton 2008; 2010; Xu 2010); predicting the trajectory of moving objects (Qiao et al. 2010); diagnosing cardiogenic heart failure and anticipating its likely evolution (Gatti, Luciani, and Stella 2011; Gatti 2011), and reasoning about complex reliability models (Cao 2011; Sturlaugson and Sheppard 2014).

While most of these applications are concerned with the most probable state of the system at certain times, we can see the advantage of moving beyond this and also estimating user-specified *values* on how the system behaves. Users may have complex valuations on how and when failures occur, intrusions are detected, diagnoses are made, etc. We introduce factored performance functions into the CTBN to support this idea.

Background

Before we describe these factored performance functions, we must formally define the CTBN, discuss the representation of the instantiations of the network, and review prior work in CTBN inference that allows a user to estimate arbitrary functions over the behavior of the network.

Continuous Time Bayesian Networks

The CTBN can be thought of as a factored Markov process. The Markov processes upon which CTBNs are built are each finite, discrete-state, continuous-time, and homogeneous with respect to time. As we will see, they are non-homogeneous with respect to states, which allows interdependence between the Markov processes and which yields conditional Markov processes. Another way of looking at the CTBN is to view it as modeling a complete system, in which each conditional Markov process is modeling a subsystem. Thus, we use "conditional Markov process" and "subsystem" interchangeably in this paper.

Formally, let $\mathbf{X}$ be a set of Markov processes $\{X_1, X_2, \ldots, X_n\}$. A continuous time Bayesian network $\mathcal{N}$ over $\mathbf{X}$ consists of two components. The first component is an initial distribution denoted $P^0_{\mathbf{X}}$ over $\mathbf{X}$ that can be specified as a Bayesian network. This distribution $P^0_{\mathbf{X}}$ is only used for determining the initial state of the process (the initial state of each X_i). The second component is a continuous-time transition model that describes the evolution of the process from its initial distribution. The transition model is specified as:

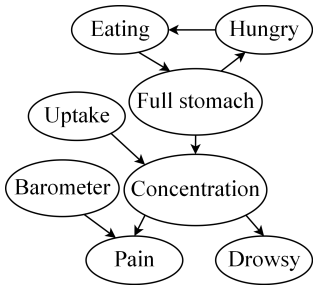

Figure 1: Example CTBN.

- A directed graph $\mathcal{G}$ in which the processes $X \in \mathbf{X}$ map to the nodes of $\mathcal{G}$,
- A set of conditional intensity matrices (CIMs) $\mathbf{A}_{X|\mathbf{Pa}(X)}$ associated with $X \in \mathbf{X}$ for each possible state instantiation of $\mathbf{Pa}(X)$, which denotes the parents of X in $\mathcal{G}$.

Each intensity matrix $\mathbf{A}_{X|\mathbf{Pa}(X)}$ is constrained such that $a_{i,j} \geq 0$ $(i \neq j)$ and $a_{i,i} \leq 0$. The entry $a_{i,j}$ $(i \neq j)$ gives the transition intensity of node X transitioning from state i to state j. The entry $a_{i,i}$ controls the amount of time that X spends in state x_i before transitioning, called the sojourn time. The sojourn times are exponentially distributed with parameter $a_{i,i}$.

Noting that the diagonal entries are non-positive, the probability density function of X to remain in state x_i is given by $|a_{i,i}| \exp(a_{i,i}t)$, with t being the amount of time spent in state x_i. In other words, the probability of remaining in state x_i decreases exponentially with respect to time. The expected sojourn time for state x_i is $1/|a_{i,i}|$. Each row is constrained to sum to zero, $\sum_j a_{i,j} = 0 \; \forall \, i$, meaning that the transition probabilities from state x_i can be calculated as $a_{i,j}/|a_{i,i}| \; \forall \, j, i \neq j$. Furthermore, for *conditional* intensity matrices, the expected sojourn times and transition probabilities of X can change depending on the states of $\mathbf{Pa}(X)$.

Figure 1 shows the example CTBN from Nodelman, Shelton, and Koller (2002) for modeling the effects of a drug on a patient. Here the nodes *Pain* and *Drowsy* will be abbreviated as *P* and *D*, respectively. Each node has two states: {*in-pain*, *pain-free*} for *P* and {*drowsy*, *non-drowsy*} for *D*.

Although the CTBN's name attempts to draw parallels between itself and Bayesian networks (BNs), the two models are fundamentally different. The nodes of a BN are conditional random variables, while the nodes of a CTBN are conditional Markov processes. Even the temporal version of a BN, the dynamic Bayesian network, still only represents state transition probabilities between discrete time-slices, whereas the CTBN encodes both state transitions and exponentially distributed sojourn times between transitions.

The graph $\mathcal{G}$ of the CTBN and the associated conditional intensity matrices define the behavior of the system as a set of interdependent subsystems. In reasoning about the system, we want to move from general defined behavior of the system to specific instances and reason about how the specific instances are evolving through time. A description of a specific instance of the system is called a trajectory and can be thought of as a data structure for recording the state transitions and their corresponding transition times. Trajectories are used for representing partial observations about the system, which can be used as evidence, and they are also used as the samples for sample-based CTBN inference algorithms.

Trajectories

Formally, let X_t be the probability distribution of the states of X at time t, and let t_s and t_e be the start time and end time of an observation, respectively, such that $t_s < t_e$. Let x be a particular state of X. The tuple $\langle t_s, t_e, x \rangle$ represents an observation of the network such that $X_t = x$ for $t_s \leq t < t_e$. The tuple could be read as "from time t_s to time t_e, the state of X was observed to be x." A trajectory of X, denoted $\sigma[X]$, is defined as a sequence of observations of X. If $t_s = 0$ for the first observation and if, for every pair of adjacent observations $\langle \langle t_s, t_e, x \rangle, \langle t'_s, t'_e, x' \rangle \rangle$ in $\sigma[X]$, $t_e = t'_s$, and $x \neq x'$, then $\sigma[X]$ is called a complete trajectory of X. Otherwise, $\sigma[X]$ is called a partial trajectory of X. A complete trajectory has no "gaps" in the observation. That is, the state of X is known from $t = 0$ until $t = t_e$ of the last observation. The full set of trajectories $\sigma[X_1] \cup \sigma[X_2] \cup \cdots \cup \sigma[X_n]$ over all of the nodes of a CTBN will be denoted as σ.

Inference

Exact inference in CTBNs can be done by combining all of the conditional intensity matrices into a single, full joint intensity matrix in which the states are the Cartesian product of the states of all of the nodes (Nodelman, Shelton, and Koller 2002). The forward-backward algorithm for Markov processes is then performed on this single matrix. Because the size of this matrix is exponential in the number of nodes, this process is infeasible for most real-world networks. Instead, we must rely on approximation algorithms for all but the simplest networks. Examples developed for CTBNs include expectation propagation (Nodelman, Koller, and Shelton 2005), importance sampling (Fan, Xu, and Shelton 2010), Gibbs sampling (El-Hay, Friedman, and Kupferman 2008), and mean-field variational methods (Cohn et al. 2009).

As mentioned earlier, sometimes the user may not be interested in querying $P(\mathbf{X}_t|\mathbf{e})$ specifically, i.e., the expected behavior of the network. Instead, the user may place different values on particular behaviors of the network and want to calculate the expected value of a given instantiation of the system. In other words, the user has a function f over the behavior the network and wants to compute the expected value of f given the evidence, $E(f|\mathbf{e})$. This query is different from (although related to) the calculation of $P(\mathbf{X}_t|\mathbf{e})$. While $\mathcal{G}$ may show X and Y to be independent in their *behavior*, there may be a dependence between X and Y in f because of how the user *values* their mutual behavior. Whereas the CTBN allows us to factor a complex system $\mathbf{X}$ into interdependent subsystems in order to tractably estimate $P(\mathbf{X}_t|\mathbf{e})$, we would like to factor f into a set of functions such that we can also tractably estimate $E(f|\mathbf{e})$.

Importance sampling is able to estimate $E(f|\mathbf{e})$ for arbitrary functions f defined over complete trajectories of the system. This algorithm serves as our underlying method as

we seek to factor f. We now briefly review importance sampling before turning to our specific contributions.

Importance Sampling

The importance sampling algorithm (Fan, Xu, and Shelton 2010), a particle-based approximate inference algorithm, takes a partial trajectory $\mathbf{e}$ as evidence and samples a proposal distribution P' that conforms to the evidence to fill in the unobserved intervals to generate a complete trajectory. Because the sampler draws from P' to force the sampling to conform to the evidence, each sample is weighted by the likelihood of the evidence, calculated as

$$w(\sigma) = \frac{P(\sigma, \mathbf{e})}{P'(\sigma)},$$

with the cumulative weight as

$$W = \sum_{\sigma \in \mathcal{S}} w(\sigma).$$

From a set of i.i.d. samples $\mathcal{S}$, we can approximate the conditional expectation of a function f given the evidence $\mathbf{e}$ as:

$$\hat{E}(f|\mathbf{e}) = \frac{1}{W} \sum_{\sigma \in \mathcal{S}} w(\sigma) f(\sigma)$$

Factored Performance Functions

A performance function measures user-specified cost/reward values over the behavior of the system. Thus, this function f can be used to represent a "global" performance function defined over the behavior of the whole system all at once. But this means that f must be passed the entire sample σ. While this allows f to be fully general and discern arbitrary behaviors of the system, this also means that the evaluation of f must be specially implemented for each function and for each network. Because the state-space of σ is exponential in the number of conditional Markov processes, simply enumerating f over all the states of network is infeasible. A representation such as a lookup table over even just the relevant states of the network would also be difficult for a user to define by hand and subsequently difficult for others to interpret. We would like to find a way to factor f to make it more manageable and understandable while retaining as much of its expressive power as possible. To do this, we move into the novel contributions of this paper, introducing performance functions local to each node and showing how to incorporate dependence in the performance functions by augmenting the structure of the CTBN with what we call performance synergy nodes.

Whereas the global performance function f places a value on the behavior of the network as a whole, we want to factor f according to the nodes in the network by assigning each node X its own performance function f_X. Each f_X is responsible for the value placed on the behavior of a single node in the network. Furthermore, f is also defined over all the transitions of a single node. While this may be useful in some cases, the value we place on certain states of a node will not be dependent on the state of the node several states ago. Therefore, we also factor each performance function with respect to time. Instead of defining $f_X(\sigma)$ over a full sample σ, we define $f_X(t_s, t_e, X_t)$ to be over the observations $\langle t_s, t_e, X_t \rangle$ in $\sigma[X]$. The performance of the entire network can now be factored as

$$f(\sigma) = \sum_{X \in \mathbf{X}} \left(\sum_{\langle t_s, t_e, X_{t_s} \rangle \in \sigma[X]} f_X(t_s, t_e, X_{t_s}) \right).$$

The factored performance function f_X is able to represent such things as fixed and variable costs over the states of X. For example, consider a performance function for some node X with states x_0 and x_1. Let $\Delta t = t_e - t_s$. Then suppose

$$f_X(t_s, t_e, X_{t_s}) = \begin{cases} c_1 + c_2 \Delta t & \text{if } X_{t_s} = x_0 \\ 0 & \text{if } X_{t_s} = x_1 \end{cases},$$

in which c_1 and c_2 are two constants representing dollars and dollars per hour, respectively, and the time is in hours. This performance function means that the system incurs a fixed cost of c_1 every time it enters state x_0 and accrues a variable cost of c_2 for every hour it remains in state x_0.

Each performance function is now responsible for calculating the performance of a single node in the network. The factorization of f also allows for greater flexibility in generating the set of samples $\mathcal{D}$. If the performance function f is defined (non-zero) for only a subset of nodes $\mathbf{X}' \subset \mathbf{X}$, the sampling algorithm only needs to record

$$\bigcup_{X \in \mathbf{X}'} \sigma[X]$$

for each sample σ, instead of every transition of every node in $\mathbf{X}$.

We can further generalize performance functions by noting that a network is not restricted to a single performance function f. We could define an entire family of performance functions $\mathcal{F} = \{f^1, f^2, \ldots, f^m\}$ for a single CTBN. Each performance function gives one "view" of the network. For example, we could define $\mathcal{F}$ to represent competing metrics, such as quantity vs. quality, and measure the trade-offs incurred by the current instance of the system. Moreover, we can evaluate $\mathcal{F}$ with a single set of samples $\mathcal{S}$, regardless of the size of $\mathcal{F}$.

Performance Synergy Nodes

The factorization of f into a single f_X for each node of the CTBN could be too restrictive for encoding the desired performance function. We now show how to augment the structure of the CTBN to make the performance functions more expressive while still preserving the factorization of f onto single nodes. First, we show how the performance function could be too restricted. Suppose that the performance functions for P and D from Figure 1 is defined as:

$$f_P(t_s, t_e, P_{t_s}) = \begin{cases} 2\Delta t & \text{if } P_{t_s} = \textit{pain-free} \\ 0 & \text{if } P_{t_s} = \textit{in-pain} \end{cases},$$

$$f_D(t_e, t_s, D_t) = \begin{cases} \Delta t & \text{if } D_{t_s} = \textit{non-drowsy} \\ 0 & \text{if } D_{t_s} = \textit{drowsy} \end{cases}.$$

In other words, we have $f = 3\Delta t$ when the *Pain* and *Drowsy* subsystems are in states *pain-free* and *non-drowsy* simultaneously. But suppose a user values being non-drowsy and pain-free at the same time twice as much as the sum of the values of being non-drowsy and pain-free separately, and the user wants the following performance function defined over P and D together to be:

$$f_{\{P,D\}}(t_s, t_e, \{P_{t_s}, D_{t_s}\}) =$$

$$\begin{cases} 6\Delta t & \text{if } P_{t_s} = \textit{pain-free} \wedge D_{t_s} = \textit{non-drowsy} \\ 2\Delta t & \text{if } P_{t_s} = \textit{pain-free} \wedge D_{t_s} = \textit{drowsy} \\ \Delta t & \text{if } P_{t_s} = \textit{in-pain} \wedge D_{t_s} = \textit{non-drowsy} \\ 0 & \text{if } P_{t_s} = \textit{in-pain} \wedge D_{t_s} = \textit{drowsy} \end{cases}$$

In this case, $f = 6\Delta t$ instead of $3\Delta t$ when *pain-free* and *non-drowsy*. The performance function $f_{P \cup D}$ does not factor into f_P and f_D as before. This introduces the concept of performance synergy between nodes. Formally, we define performance synergy to be the case in which, for two nodes X and Y, their joint performance function $f_{\{X,Y\}}$ cannot be factored into f_X and f_Y such that, for all $x \in X$ and $y \in Y$,

$$f_{\{X,Y\}}(\{x, y\}) = f_X(x) + f_Y(y).$$

Suppose, on the other hand, that $f_{\{X,Y\}}$ is able to be factored partially into f_X and f_Y such that the above equality holds for at least one state $x \in X$ and one state $y \in Y$. Then all other states $x' \in X$ and $y' \in Y$ for which the equality does not hold exhibit either positive or negative performance synergy.

Individual states $x \in X$ and $y \in Y$ exhibit positive performance synergy if and only if

$$f_{\{X,Y\}}(\{x, y\}) > f_X(x) + f_Y(y).$$

Likewise, individual states $x \in X$ and $y \in Y$ exhibit negative performance synergy if and only if

$$f_{X \cup Y}(x \cup y) < f_X(x) + f_Y(y).$$

Note that the synergy introduced here is *not* in terms of probabilities, as in additive and product synergy of qualitative probabilistic networks (Druzdzel and Henrion 1993), but in terms of performance functions. Performance synergy implies that the performance of multiple nodes is dependent. Synergy occurs at the state level, and nodes can exhibit both positive and negative synergy at the same time. To account for synergy in the performance functions while maintaining the factorization of f, we add performance synergy nodes into the network. A synergy node is set as the child of all nodes contributing to the synergy. For the P and D synergy example, we add a synergy node PD^+ (denoting positive synergy) as a child of P and D. The augmented section of the network is shown in Figure 2, and the conditional intensity matrices for PD^+ are given in Table 1. The combination of ∞ and 0 force the synergy node to transition immediately to the active (inactive) state and remain there for as long as the logical expression is satisfied (unsatisfied). In practice, ∞ is simulated by a sufficiently large value that allows near-instantaneous transitions, as compared to the other transitions times. Finally, we set the performance function of PD^+ as

$A_{PD^+ \mid P_{t_s}=\textit{pain-free}, D_{t_s}=\textit{non-drowsy}}$		
	inactive	*active*
inactive	$-\infty$	∞
active	0	0

$A_{PD^+ \mid P_{t_s}=\textit{pain-free}, D_{t_s}=\textit{drowsy}}$ $A_{PD^+ \mid P_{t_s}=\textit{in-pain}, D_{t_s}=\textit{non-drowsy}}$ $A_{PD^+ \mid P_{t_s}=\textit{in-pain}, D_{t_s}=\textit{drowsy}}$		
	inactive	*active*
inactive	0	0
active	∞	$-\infty$

Table 1: Conditional intensity matrices of PD^+.

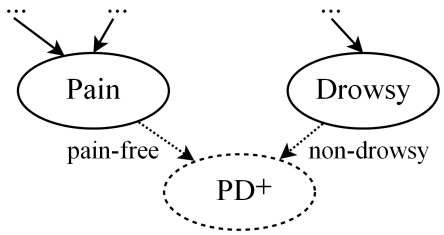

Figure 2: Positive synergy node for *pain-free* and *non-drowsy*.

$$f_{PD^+}(t_s, t_e, PD^+_{t_s}) = \begin{cases} 3\Delta t & \text{if } PD^+_{t_s} = \textit{active} \\ 0 & \text{if } PD^+_{t_s} = \textit{inactive} \end{cases}$$

Borrowing the idea from (Cao 2011), the CIMs of PD^+ act as a logical expression over the states of the parents P and D. Whenever P is *pain-free* and D is *non-drowsy*, the synergy node PD^+ immediately switches to *active* and yields an additional $3\Delta t$ in performance. This yields the desired performance function with the factorization as $f = f_P + f_D + f_{PD^+}$. Thus, f is still factored onto individual nodes. Furthermore, the synergy node PD^+ provides a graphical representation of the performance function. We can see at a glance in Figure 2 that the performance functions of P and D are dependent. Furthermore, we can define the synergy for *pain-free* and *non-drowsy* as its own value, instead of having to define it in the function $f_{\{P,D\}}$, in which the synergistic effect of *pain-free* and *non-drowsy* is less obvious.

Experiments

We demonstrate the use of the synergy node concept on a real-world reliability model adapted from (Cao 2011) that describes the uptime of a vehicle system. The model, shown in Figure 3, consists of three subsystems: chassis (*CH*), powertrain (*PT*), and electrical (*EL*). The chassis is comprised of four components, each having their own failure and repair rates: suspension (SU), brakes (*BR*), wheels and tires (*WT*), and axles (*AX*). Likewise, the powertrain subsystem is comprised of three subsystems: cooling (*CO*), engine (*EG*), and transmission (*TR*).

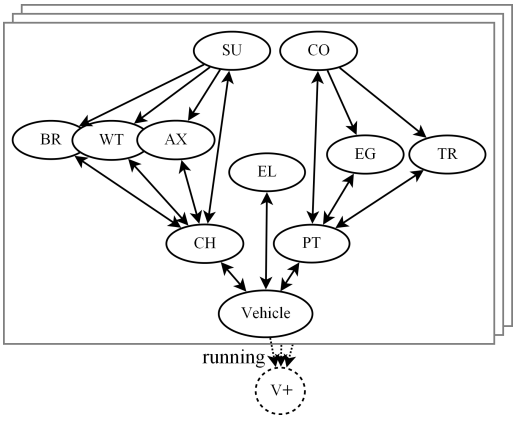

Figure 3: CTBN for fleet of vehicles with synergy node.

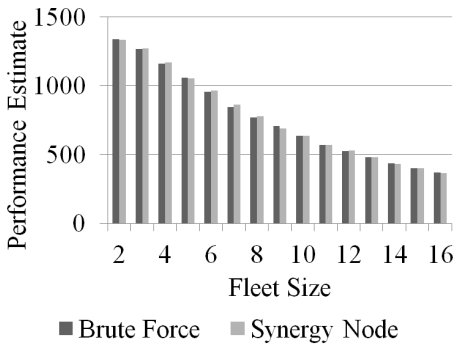

Figure 4: Performance estimates of the brute force and synergy node approaches.

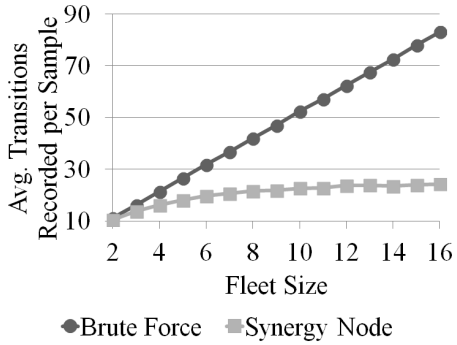

Figure 5: Average number of transitions per sample of the brute force and synergy node approaches.

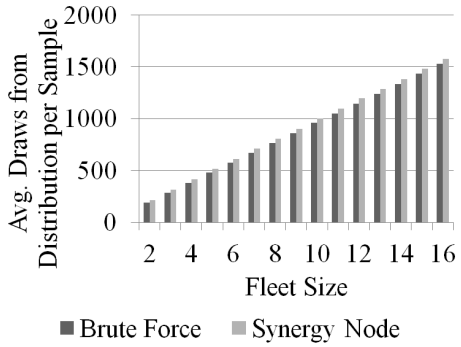

Figure 6: Average number of draws from an exponential or multinomial distribution per sample of the brute force and synergy node approaches.

For our experiments, we have a fleet of vehicles. Each vehicle model can incorporate its own evidence, e.g., repair and usage history. We want to calculate a synergistic measure of performance across the entire fleet, represented in the node V^+. We compare the use of the synergy node with the brute force approach, i.e., of evaluating a performance function defined over all of the *Vehicle* nodes at once. Therefore, the network for the brute force approach does not include the V^+ node. For the synergy node approach, the V^+ node becomes *active* when all vehicles are running, otherwise it remains *inactive*. Suppose that the performance function for V^+ is defined as

$$f_{V^+}(t_s, t_e, V^+_{t_s}) = \begin{cases} \Delta t & \text{if } V^+_{t_s} = \textit{active} \wedge \Delta t \geq 40 \\ 0 & \text{if otherwise} \end{cases}$$

In other words, additional performance is gained when all of the vehicles are running simultaneously for at least 40 hours. The performance gained is proportional to the amount of time that all of the vehicles are running until the next repair.

We varied the fleet size from 2 to 16 vehicles and queried the expected performance over 2000 hours of operation starting with all vehicles in running condition. We simulated ∞ in the CIMs of V^+ as 10^{10}. We used importance sampling to generate 10,000 samples for each fleet size and for the brute force and synergy node approaches. We compared accuracy of the performance estimate, average number of transitions, and average number of times the sampler must draw from an exponential or multinomial distribution. Because we only needed to save the trajectories for the *Vehicle* nodes, the average number of transitions per sample dictates how many times the performance function must be evaluated. Because each sample is a series of sojourn times and transitions, the number of times the sampler draws from an exponential or multinomial distribution is the driving factor in the complexity of creating the samples.

The performance estimates of the brute force approach and the synergy node approach is shown in Figure 4. As the graph shows, the synergy node is able to return an estimate consistent with the brute force approach. The average relative error between the two approaches is less than 1%.

The average number of transitions per sample of the brute force approach and the synergy node approach is shown in Figure 5. Note that we did not record transitions for all of the nodes, only the nodes contributing to the performance function. In other words, the brute force approach used $\sigma = \cup_i \sigma[\textit{Vehicle}_i]$ for fleet size i, while the synergy node approach used $\sigma = \sigma[V^+]$. As the graph shows, the number of transitions increases linearly for the brute force approach, as expected. For the synergy node approach, on the

other hand, the curve behaves logarithmically. This is because, as the number of vehicles increases, the proportion of time that all are running simultaneously decreases. Therefore, the number of transitions between the states of the synergy node decreases. This also means that the sample path for the synergy node takes fewer evaluations to estimate the performance.

Finally, the average number of times the sampler must draw from a distribution is shown in Figure 6. As the graph shows, the addition of the synergy node does increase the complexity of generating each sample; however, the complexity is not greatly increased, as the curve suggests only a small, constant-factor increase.

Conclusions

In this paper, we formalize factored performance functions for the CTBN. Existing CTBN inference algorithms support estimating arbitrary functions over the behavior of the network, but by factoring these functions onto the nodes of the network, we can achieve a representation of performance that we argue is more easily understood and implemented. Furthermore, to support more complex interactions in which the performance function cannot be factored in a straightforward manner, we show how to maintain a factorization by augmenting the structure of the CTBN with synergy nodes. We argue that such complex performance information is more easily understood in terms of the synergistic relationship between nodes. We show a real-world example in which the synergy node is able to capture the performance evaluation between multiple nodes without a significant increase in complexity and without degrading accuracy.

As future work, we intend to demonstrate other scenarios that use synergy nodes, as well as families of performance functions, on a wider variety of real-world networks.

References

Cao, D. 2011. *Novel models and algorithms for systems reliability modeling and optimization*. Ph.D. Dissertation, Wayne State University.

Cohn, I.; El-Hay, T.; Friedman, N.; and Kupferman, R. 2009. Mean field variational approximation for continuous-time Bayesian networks. In *Proceedings of the Twenty-Fifth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-09)*, 91–100. Corvallis, Oregon: AUAI Press.

Druzdzel, M., and Henrion, M. 1993. Efficient reasoning in qualitative probabilistic networks. In *In Proceedings of the 11th National Conference on Artificial Intelligence (AAAI–93*, 548–553.

El-Hay, T.; Friedman, N.; and Kupferman, R. 2008. Gibbs sampling in factorized continuous-time Markov processes. In *Proceedings of the Twenty-Fourth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-08)*, 169–178. Corvallis, Oregon: AUAI Press.

Fan, Y., and Shelton, C. 2009. Learning continuous-time social network dynamics. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, 161–168.

Fan, Y.; Xu, J.; and Shelton, C. R. 2010. Importance sampling for continuous time Bayesian networks. *The Journal of Machine Learning Research* 99:2115–2140.

Gatti, E.; Luciani, D.; and Stella, F. 2011. A continuous time Bayesian network model for cardiogenic heart failure. *Flexible Services and Manufacturing Journal* 1–20.

Gatti, E. 2011. *Graphical models for continuous time inference and decision making*. Ph.D. Dissertation, Università degli Studi di Milano-Bicocca.

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

Murphy, K. P. 2002. *Dynamic Bayesian networks: representation, inference and learning*. Ph.D. Dissertation, University of California.

Ng, B.; Pfeffer, A.; and Dearden, R. 2005. Continuous time particle filtering. In *International Joint Conference on Artificial Intelligence*, volume 19, 1360.

Nodelman, U., and Horvitz, E. 2003. Continuous time Bayesian networks for inferring users' presence and activities with extensions for modeling and evaluation. Technical report.

Nodelman, U.; Koller, D.; and Shelton, C. 2005. Expectation propagation for continuous time Bayesian networks. In *Proceedings of the Twenty-First Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-05)*, 431–440. Arlington, Virginia: AUAI Press.

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

Qiao, S.; Tang, C.; Jin, H.; Long, T.; Dai, S.; Ku, Y.; and Chau, M. 2010. PutMode: prediction of uncertain trajectories in moving objects databases. *Applied Intelligence* 33(3):370–386.

Shi, D.; Tang, X.; and You, J. 2010. An intelligent system based on adaptive CTBN for uncertainty reasoning in sensor networks. *Intelligent Automation & Soft Computing* 16(3):337–351.

Sturlaugson, L., and Sheppard, J. W. 2014. Sensitivity analysis of continuous time Bayesian network reliability models. Submitted to *SIAM/ASA Journal on Uncertainty Quantification*.

Xu, J., and Shelton, C. 2008. Continuous Time Bayesian Networks for Host Level Network Intrusion Detection. In Daelemans, W.; Goethals, B.; and Morik, K., eds., *Machine Learning and Knowledge Discovery in Databases*, volume 5212 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. 613–627.

Xu, J., and Shelton, C. 2010. Intrusion detection using continuous time Bayesian networks. *J. Artif. Int. Res.* 39(1):745–774.

Xu, J. 2010. *A Continuous Time Bayesian Network Approach for Intrusion Detection*. Ph.D. Dissertation, University of California.