Journal of Applied Logic 22 (2017) 28–45

Contents lists available at ScienceDirect

Journal of Applied Logic

www.elsevier.com/locate/jal

Factored performance functions and decision making in continuous time Bayesian networks

Liessman Sturlaugson, Logan Perreault *, John W. Sheppard

Computer Science Department, Montana State University, Bozeman, MT 59717, United States

ARTICLE INFO

Article history:
Available online 15 November 2016

Keywords:
Continuous time Bayesian network
Performance function
Synergy
Multi-objective optimization

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 changes 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. We introduce the notion of optimization for CTBNs, and show how a family of performance functions can be used as the evaluation criteria for a multi-objective optimization procedure.

1. 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 [14] 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 [11].

* Corresponding author.
E-mail address: perreault.logan@gmail.com (L. Perreault).

http://dx.doi.org/10.1016/j.jal.2016.11.030
1570-8683/

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 [13]; robot monitoring [12]; modeling server farm failures [9]; modeling social network dynamics [5]; modeling sensor networks [17]; building intrusion detection systems [21–23]; predicting the trajectory of moving objects [16]; diagnosing cardiogenic heart failure and anticipating its likely evolution [7,8], and reasoning about complex reliability models [1,20].

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. In this extended version of our conference paper [18], we introduce families of factored performance functions into the CTBN to support this idea. We then formalize the concept of optimization in CTBNs, and show how families of performance functions can be used as the evaluation criteria for a multi-objective optimization problem. We demonstrate the use of factored performance functions with three networks. In the first network, we show how factoring the performance function can increase the efficiency of inference without loss in accuracy. The next two networks demonstrate how families of factored performance functions can be used for CTBN multi-objective optimization.

2. Background

Before we describe these factored performance functions, we must formally define the BN and then the CTBN. We then 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.

2.1. Bayesian networks

Bayesian networks are probabilistic models corresponding to joint probability distributions that utilize conditional dependencies among random variables. Bayesian networks are used in a wide variety of domains, such as image processing, search, information retrieval, diagnostics, and many others. A Bayesian network uses observations, or evidence, and previously determined conditional probabilities to give the probability of a certain state.

Definition 1 *(Bayesian network).* A Bayesian network $\mathcal{B}$ is a directed, acyclic graph whose vertices correspond to random variables of a distribution, and the edges correspond to conditional dependencies between random variables. Each vertex has an associated conditional probability distribution, denoted $P(X_i|\mathrm{Pa}(X_i))$, where $\mathrm{Pa}(X_i)$ are the direct predecessors (also called "parents") of vertex X_i.

Bayesian networks are a way of representing joint probability distributions in a more compact way by using conditional dependencies among the random variables. Instead of needing to enumerate the entire joint probability distribution we can just use the product rule from probability to get the following:

$$P(X_1, ..., X_n) = P(X_1) \prod_{i=2}^{n} P(X_i|X_1, ..., X_{i-1})$$

Bayesian networks are able to exploit conditional independence, which is represented in the directed acyclic graph $\mathcal{G}$, to reduce the model's complexity and yield the following:

$$P(X_1, ..., X_n) = \prod_{i=1}^{n} P(X_i|\mathrm{Pa}(X_i)).$$

Bayesian networks are frequently used because the representation they use is often easier to understand than other graphical models. They can be much easier to tell what a particular network is representing and how it will behave in the presence of evidence.

2.2. 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 X denote a Markov process. For simplicity of notation, we also use X to denote the set of its states. Let $|X|$ denote the number of states in X. Let $x \in X$ denote state x of Markov process X, and let $x_i \in X$ for $i \in [1, \ldots, |X|]$ denote the ith state of Markov process X. Let each X_i where $i \in [1, \ldots, n]$ denote a finite-state Markov process, and let $\mathbf{X}$ be a set of Markov processes $\{X_1, X_2, \ldots, X_n\}$.

Definition 2 *(CTBN).* A CTBN $\mathcal{N}$ over $\mathbf{X}$ consists of two components. The first component is an initial distribution over $\mathbf{X}$, denoted $P^0_{\mathbf{X}}$, that can be specified as a Bayesian network (BN). 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:

- A directed graph $\mathcal{G}$ consisting of a node for each Markov process $X \in \mathbf{X}$.
- 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 $\mathcal{G}$.

Let $a_{i,j}$ denote the entry of $\mathbf{A}_{X|\mathbf{Pa}(X)}$ at row i and column j. 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 $|a_{i,i}|e^{a_{i,i}t}$, 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 $\mathbf{Pa}(X)$.

Note that the user must specify the unit of time for t. The CTBN model and its inference algorithms make no assumption on the unit of time, but the unit should be chosen to avoid both numeric overflow and underflow.

Fig. 1 shows the example CTBN from Nodelman, Shelton and Koller [14] for modeling the effects of a drug on a patient. Here the nodes *Pain* and *Drowsy* will be abbreviated as P and D, respectively. The nodes P and D both have 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 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

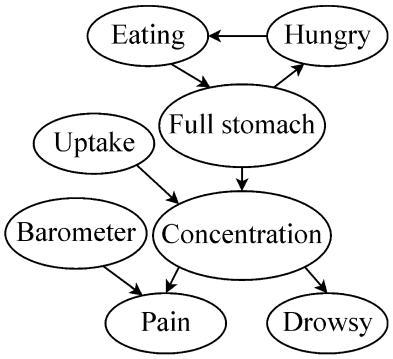

Fig. 1. Example CTBN drug network.

state transitions and exponentially distributed sojourn times between transitions. Another key difference is that the graph of a BN is constrained to be a directed acyclic graph, whereas the graph of a CTBN is simply a directed graph without the acyclic constraint.

The directed graph $\mathcal{G}$ of the CTBN and the associated CIMs 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.

2.3. Trajectories

Formally, let $X(t)$ be the state 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."

Definition 3 *(Trajectory).* A trajectory of X, denoted $\sigma[X]$, is defined as a finite 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. A partial trajectory used as evidence is denoted $\mathbf{e}$.

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.

2.4. Inference

Exact inference in CTBNs can be done by combining all of the CIMs into a single, full joint intensity matrix in which the states are the Cartesian product of the states of all of the nodes [14]. 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 [15], importance sampling [6], Gibbs sampling [4], and mean-field variational methods [2].

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.

Definition 4 *(Expected performance).* Let $f : \sigma \to \mathbb{R}$ be a function on the behavior of a CTBN $\mathcal{N}$. Let $\mathcal{S}_{\mathbf{e}}$ denote the set of all complete trajectories that conform to evidence $\mathbf{e}$. Let $P(\sigma)$ denote the probability of trajectory σ. The expected performance of $\mathcal{N}$ is defined as:

$$E(f|\mathbf{e}) = \sum_{\sigma \in \mathcal{S}_{\mathbf{e}}} P(\sigma) f(\sigma).$$

This query is different from (although related to) the calculation of the probability of a given states of X at time t given evidence $\mathbf{e}$, $P(\mathbf{X}(t)|\mathbf{e})$. 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.

2.5. Importance sampling

The importance sampling algorithm [6], 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. The sampler generates a set of independent and identically distributed samples σ (complete trajectories) $\mathcal{S}_{\mathbf{e}}$. 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

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

where $P(\sigma, \mathbf{e})$ is the probability of the sampler generating σ and $\mathbf{e}$. We calculate a normalizer as the cumulative weight:

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

We can approximate the conditional expectation of a function f given the evidence $\mathbf{e}$ as:

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

In general, inference over CTBNs, including calculating the performance functions introduced here, is NP-hard for both exact and approximate inference [19]. While this is true in the general case, we are interested in networks for which inference is tractable.

3. 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 $\mathcal{F}$.

4. 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 Fig. 1 are 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.

Definition 5 *(Performance synergy).* Performance synergy is the case in which, for two nodes X and Y, with joint performance function $f_{\{X,Y\}}$, there do not exist functions 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.

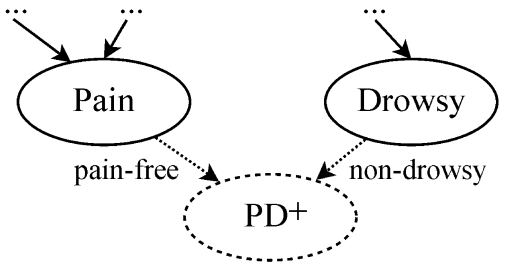

Fig. 2. Positive synergy node PD^+ for *pain-free* and *non-drowsy*.

Table 1
Conditional intensity matrices of PD^+.

$A_{PD^+\|P(t_s)=pain\text{-}free, D(t_s)=non\text{-}drowsy}$		
	inactive	*active*
inactive	$-\infty$	∞
active	0	0

$A_{PD^+\|P(t_s)=pain\text{-}free, D(t_s)=drowsy}$ $A_{PD^+\|P(t_s)=in\text{-}pain, D(t_s)=non\text{-}drowsy}$ $A_{PD^+\|P(t_s)=in\text{-}pain, D(t_s)=drowsy}$		
	inactive	*active*
inactive	0	0
active	∞	$-\infty$

Definition 6 *(Positive performance synergy).* Positive performance synergy is the case in which, for individual states $x \in X$ and $y \in Y$,

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

Definition 7 *(Negative performance synergy).* Negative performance synergy is the case in which, for individual states $x \in X$ and $y \in Y$,

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

Note that the synergy introduced here is *not* in terms of probabilities, as in additive and product synergy of qualitative probabilistic networks [3], 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.

Definition 8 *(Synergy node).* A synergy node X is a two-state child of a subset of CTBN nodes that exhibit either positive or negative performance synergy when in state combination S. The two states of X are *active* and *inactive*. The CIMs of X deterministically set the state of X to *active* whenever the parents are in states S and to *inactive* otherwise. The performance function for X incorporates the performance synergy of its parents when X is in *active*, while it is 0 when X is in *inactive*. Evidence is never applied directly to synergy nodes.

We can demonstrate the utility of synergy nodes by again considering the P and D synergy example. In this case, 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 Fig. 2, and the CIMs 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

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

Borrowing the idea from [1], 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 Fig. 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.

5. Multi-objective optimization in CTBNs

With some models, we may be interested in evaluating the expected performance of the associated system. For example, in the drug effect network, we can measure the expected amount of discomfort to the patient under different circumstances. With other models, we may have some amount of control over the state of the system. In the drug effect network, we can control the dosage of the drug and the time it is administered. In such cases, we might be interested in computing those actions that optimize the performance function. We can also place constraints on the possible actions and timing of the actions to ensure that the events that occur are feasible.

Definition 9 *(Control nodes).* Let $\mathcal{N}$ be a CTBN consisting of a set of Markov processes $\mathbf{X}$. Let f be a performance function defined over $\mathcal{N}$. Control nodes $\mathbf{Y} \subseteq \mathbf{X}$ are nodes for which the user can specify $Y(t)$ for $Y \in \mathbf{Y}$ subject to constraints C.

The control nodes allow user actions on the system to be represented in the CTBN, and for the user to optimize the performance functions over the set of possible actions. This is done by specifying $\sigma[\mathbf{Y}]$ as evidence for $E(f|\mathbf{e})$. Let $\mathcal{S}_C$ be the set of all complete trajectories that conform to the constraints C. The problem of CTBN optimization attempts to find a complete trajectory $\sigma[\mathbf{Y}] \in \mathcal{S}_C$ that maximizes the expected value:

$$\operatorname*{argmax}_{\sigma[\mathbf{Y}] \in \mathcal{S}_C} E(f|\sigma[\mathbf{Y}]).$$

For multi-objective optimization, we optimize over a family of m user-defined performance functions $\mathcal{F} = \{f^1, \ldots, f^m\}$,

$$\operatorname*{argmax}_{\sigma[\mathbf{Y}] \in \mathcal{S}_C} \left(E(f^1|\sigma[\mathbf{Y}]), E(f^2|\sigma[\mathbf{Y}]), \ldots, E(f^m|\sigma[\mathbf{Y}])\right).$$

As is often the case with multi-objective optimization, when all of the functions of $\mathcal{F}$ cannot be combined into a single value that can be optimized, we are interested in estimating the Pareto frontier [10]. In this context, the Pareto frontier is the set of solutions where an improvement for any value of a performance function f^i implies a degradation in the value of some other performance function f_j.

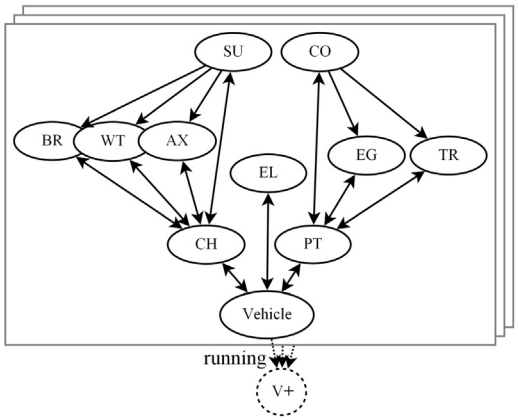

Fig. 3. CTBN for fleet of vehicles with synergy node.

The constraints on $\mathbf{Y}$ are used to define the *allowable* trajectories. The constraints are necessary to prevent infeasible actions, but the constraints can also be used to make the optimization problem more tractable. The action trajectories are defined over continuous time, so the user may specify discrete time intervals over which the action holds. The optimization can then enumerate these discrete choices rather than optimizing over the entire continuum.

6. Experiments

To begin, the Vehicle Fleet Performance experiment in section 6.1 is used to demonstrate the behavior of factored performance functions as compared to defining functions over the entire network. We validate the correctness of using factored performance functions by comparing the estimated performance values to those obtained using a brute force approach. To identify the effect that factored performance functions have on computational complexity, statistics are recorded for how many distribution draws are made to generate the samples during inference and how many transitions are recorded for the samples. The complexity of the factored and brute force approaches are measured and compared in terms of these distribution draws and recorded transitions.

Next, we demonstrate how CTBNs can be used to perform multi-objective optimization over a family of performance functions. The Drug Optimization experiment in section 6.2 defines two performance functions intended to reflect a patient's comfort level and potential weight gain that occur as a result from taking a drug. The number of doses taken by the patient per day is varied in an attempt to find an amount that maximizes comfort level while minimizing weight gain. The Vehicle Fleet Optimization experiment in section 6.3 also defines two performance functions, but instead focuses on vehicle uptime and maintenance cost. Here the goal is to assign an optimal set of technicians to maximize vehicle uptime and minimize cost.

6.1. Vehicle Fleet Performance

We demonstrate the use of the synergy node concept on a real-world reliability model adapted from [1] that describes the uptime of a vehicle system. The model, shown in Fig. 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).

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,

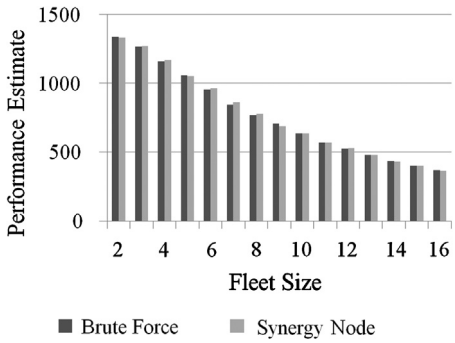

Fig. 4. Performance estimates of the brute force and synergy node approaches.

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

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

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 Fig. 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 Fig. 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 $\sigma = \cup_i \sigma[\mathit{Vehicle}_i]$ 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 Fig. 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.

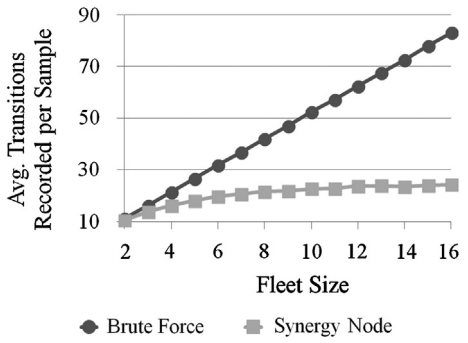

Fig. 5. Average number of transitions per sample of the brute force and synergy node approaches.

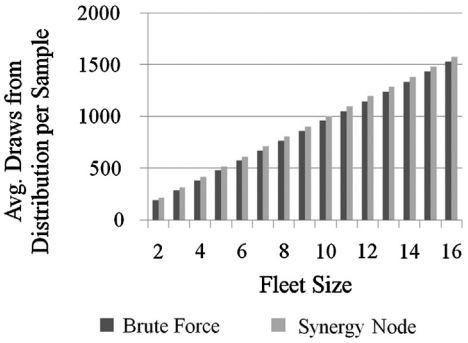

Fig. 6. Average number of draws from an exponential or multinomial distribution per sample of the brute force and synergy node approaches.

This experiment shows an example in which the factored performance function can increase efficiency of inference over a brute force approach. This increase is not guaranteed for all networks, but factoring the performance function has the potential to do so. Furthermore, factored performance functions are simpler to define (versus functions defined over the entire state-space), and synergy nodes offer a graphical way of representing and reasoning over more complex performance interactions among sets of nodes.

6.2. Drug optimization

For this experiment, we work with a modified version of the drug effect network shown in Fig. 7. The original drug effect network, shown in Fig. 1, was presented by Nodelman, Shelton and Koller [14] and was designed to monitor the effect of a drug that had been applied to a patient at some previous time. We modified the network by adding a Drug node, which is set to be the parent of the existing Uptake node. In the original model, Uptake transitions from state u_1 to state u_0 with an intensity of 0.5 and remains in state u_0 for the remainder of the process. With the addition of the two-state Drug node, this behavior remains true when Drug is in state d_0, but Uptake transitions to u_1 again with an intensity of ∞ when Drug is in state d_1. Furthermore, the initial distribution for Uptake is adjusted so that the variable deterministically starts in u_0. Conceptually this means that uptake will not occur unless the drug is applied, at which point it occurs instantaneously. After the drug is not longer applied (Drug returns to d_0), the uptake stops after a time that is distributed exponentially with a rate of 0.5. This behavior allows us to model the application of the drug directly, rather than assuming the event occurred at some previous time. Since the application of the drug is the variable we are able to control in the model, we consider it to be a member of the optimization set, which is denoted using shaded nodes in the figure.

In addition to the Drug node, the model is also modified to account for the effect that a drug may have on hunger. Some drugs affect appetite, and to capture this behavior we have added a dependency arc from

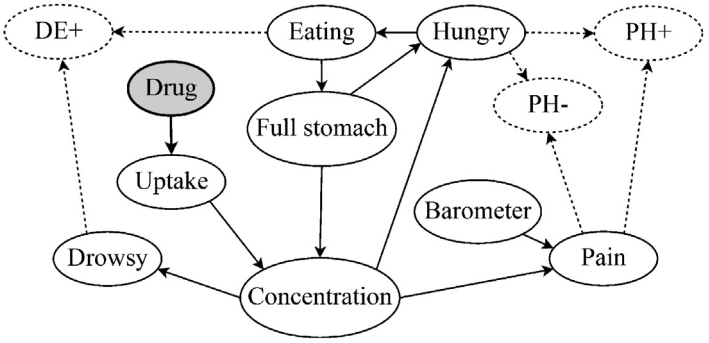

Fig. 7. Drug network modified to investigate how application of the drug affects comfort and weight gain.

the three-state Concentration node to the two-state Hungry node in the original network. To account for the drug's effect on appetite, the first rows of the Hungry CIMs are multiplied by a factor of ϕ_0, ϕ_1, or ϕ_2, depending on if Concentration is in state c_0, c_1, or c_2, respectively. Similarly, the second rows of the Hungry CIMs are multiplied by a factor of $\frac{1}{\phi_0}$, $\frac{1}{\phi_1}$, or $\frac{1}{\phi_2}$, depending on the state of Concentration. We set $\phi_0 = 1.0$, so that when Concentration is in state c_0, Hungry behaves the same as in the original drug effect network. We set $\phi_1 = 5.0$ and $\phi_2 = 10.0$, so that as the concentration of the drug increases, the intensity for transitioning into the h_1 state increases while transitioning back to state h_0 decreases. This effectively models the scenario where the drug causes the patient to feel hungrier. By setting ϕ_1 and ϕ_2 to some constant $c < 1.0$, we could have also modeled loss of appetite as a side effect.

Ultimately we would like to examine both a patient's comfort level and their potential for weight gain due to the application of the drug. To model these factors of interest, we develop two performance functions that we factor in the network. First, we define a function that describes patient comfort in terms of the pain experienced by the patient and how hungry the patient is. We consider both positive and negative contributions to the Comfort function, which we refer to as benefits and penalties, respectively. In an attempt to capture the instant gratification received by a patient, benefits are awarded when either Pain or Hungry transition back to a state of p_0 or h_0. In addition, a constant benefit is also awarded for every hour the patient remains in either the p_0 or h_0 state. In a similar fashion, penalties are assigned whenever a transition is made into the states p_1 or h_1, and a constant negative contribution to the function is dispensed for every hour spent in either one of these states.

To model the notion that a patient values the state of both Hungry and Pain more than the linear combination of the two, we also monitor additional synergy nodes $PH+$ and $PH-$ added to the network, and shown as nodes with a dotted border in the figure. $PH+$ is a positive synergy node that models the case where the patient is neither hungry nor in pain, while $PH-$ is a negative synergy node that represents the case when the patient is both hungry and in pain. This accounts for any additional pleasure or displeasure that a patient feels when multiple things are going right or going wrong.

The potential that the drug has to induce weight gain for a patient is described by the WeightGain performance function. We define this function in terms of the Eating and Drowsy nodes, with the notion being that the calorie intake will be higher if the patient eats more, and the calories burnt will be lower when the patient is drowsy for longer periods of time. While immediate pleasure or displeasure was attained for the comfort variable upon entering a state, the concept of weight gain has no corresponding instantaneous behavior. For example, while *being* drowsy may cause weight gain, *becoming* drowsy does not. In this case we have a more simplistic WeightGain performance function that simply adds a positive contribution for every hour spent in the Eating $= e_1$ or Drowsy $= d_1$ states. Here again, we account for the additional contribution for both eating and being drowsy using the synergy node $DE+$, which contributes an additional positive value to WeightGain not already captured by Eating and Drowsy individually.

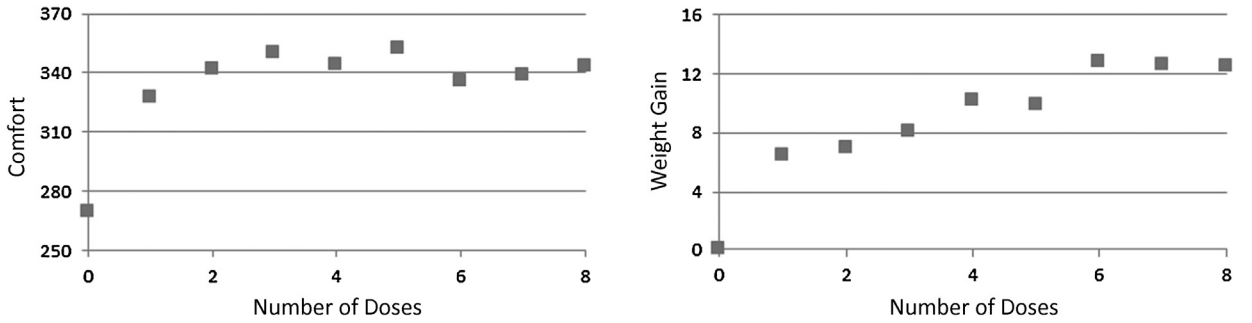

Fig. 8. Performance estimates of Comfort (left) and WeightGain (right) as the number of doses is increased.

The goal of this multi-objective optimization problem is to find a trajectory for the Drug node that maximizes Comfort while minimizing WeightGain. We first constrain the possible set of trajectories to eliminate behavior that is not possible or does not align with our model. In this case, application of the drug is intended to be an event that occurs at a specific time. We can simulate this by ensuring that upon transitioning to state d_1, the variable transitions back to state d_0 after some short time interval ϵ. For our experiments, we set ϵ to 0.001. When the drug node is applied (D enters state d_1), Uptake instantly transitions to state u_1 due to the infinite transition intensity. When D transitions back to state d_0 ϵ time later, Uptake begins its normal behavior of transitioning back to u_0 with rate 0.5.

While the described constraint ensures only valid trajectories are considered, we are still left with the intractable problem of evaluating all possible times at which to transition into state d_1. To overcome this problem, we can apply domain knowledge to constrain the trajectories further to the options we expect to be most feasible. In this case, we know that the drug should be administered at routine intervals. These constraints reduce the problem of evaluating all possible transition times to the more tractable problem of identifying the best time between drug dose applications.

For this experiment, we are interested in the effect of drug applications administered throughout a day, and therefore run inference from time $t_s = 0$ to time $t_e = 12$ hours. Identifying the optimal time between dose applications is equivalent to the problem of identifying the number of doses administered per day. We begin by evaluating the case where no doses are applied, which equates to a time between doses greater than 12. We then increase the doses per day from 1 to 8, applying evidence to the Drug node that forces a transition to and from state d_1 at routine intervals throughout the day. At 8 doses per day, this equates to an application event occurring every 1.5 hours.

The estimated values for Comfort and WeightGain as a function of the number of applied doses are shown in Fig. 8. The Comfort level for the patient increases as the number of doses increases, but the payoff levels off after two or three applications of the drug. Any number of doses above three remains at a Comfort level of approximately 340. The same positive increase can be observed for WeightGain, but the trend continues until hitting the maximum of eight doses. Optimizing either function independently is a simple matter of choosing the largest or smallest value. The task of optimizing both functions simultaneously is more difficult.

To identify the optimal choice for both performance functions, Fig. 9 plots WeightGain as a function of Comfort. The goal is to maximize comfort and minimize WeightGain, so an ideal datapoint is located in the bottom right of the graph. Here we see that 0 drug applications minimizes the expected WeightGain, while 5 drug applications maximizes Comfort. The darker nodes make up the Pareto frontier, which strictly dominate the performance of the lighter nodes. The dominated nodes occur due to the ceiling for the Comfort values that occurs at around four doses, which is not shared by the WeightGain performance function. Any of the darker nodes in the Pareto frontier are a valid choice for the optimal drug applications depending on how much Comfort is valued as compared to WeightGain. In this case, two or three doses appear to provide a good balance between Comfort and WeightGain.

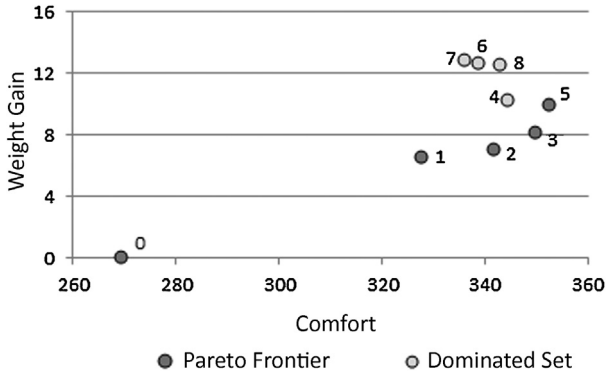

Fig. 9. Performance estimates of WeightGain displayed as a function of Comfort. Each datapoint is labeled with the number of dose applications.

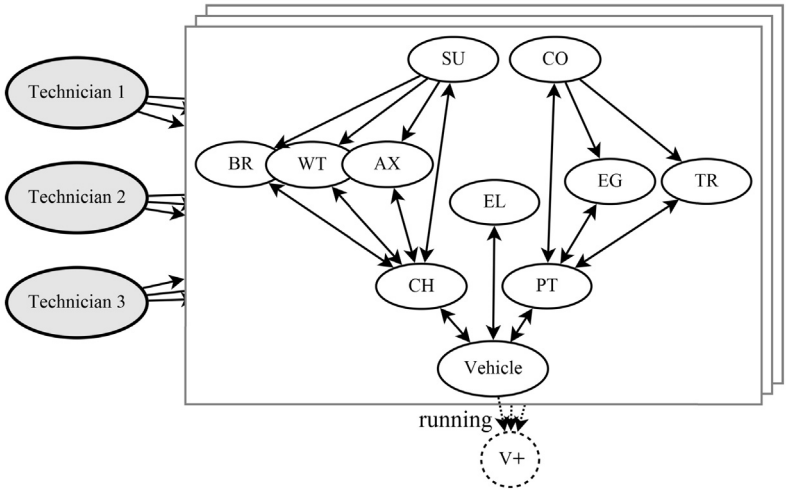

Fig. 10. Vehicle network modified to include technicians.

6.3. Vehicle fleet optimization

We now shift our attention back to the vehicle fleet model used in the experiment from section 6.1. The fleet size is set to five, meaning that there are five separate replicated models of the vehicle network that are each connected via the synergy node. To make the model suitable for optimization, we add the notion of specialized technicians who are capable of maintaining and repairing the vehicles in the fleet. We include three technician nodes in the network, which are added to the parent set of all remaining nodes in the network. This means that the behavior of all components in each vehicle is affected by the behavior of the technician nodes. The intent is for these to provided added expertise in maintaining the vehicles, beyond that already covered by the model. The modified vehicle fleet model is shown in Fig. 10.

The technician nodes have two states corresponding to the absence or presence of the technician. The presence of a technician decreases the time to repair, and increases the expected time to failure for a component due to preventative maintenance. This equates to increasing the intensities for transitioning back to state 0, while decreasing the intensities for transitioning away from state 0. Technician 1 and Technician 2 are considered to be Level I technicians, and scale the repair rates by a factor of 1.5, and the failure rates by a factor of 0.8. Technician 3 is intended to model a Level II technician, and therefore has a stronger influence on the repair and failure rates. In this case, the failure rate is reduced by a factor of 0.5 and the repair rate is increased by a factor of 2.0.

Table 2
Repair costs for vehicle components.

Component	Repair cost
BR (Brakes)	950
WT (Wheels/Tires)	700
AX (Axels)	2000
SU (Suspension)	850
EG (Engine)	450
TR (Transmission)	6500
CO (Cooling)	200

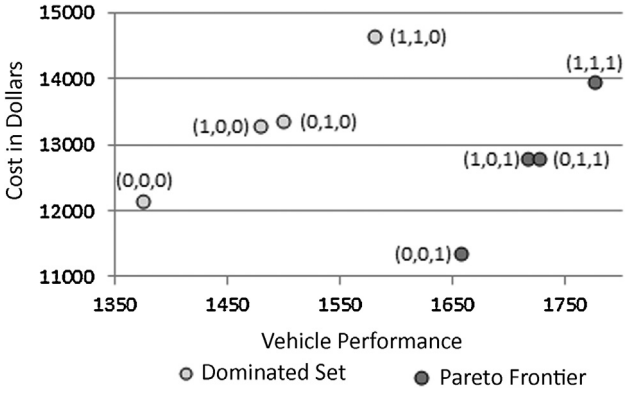

Fig. 11. Performance estimates of Cost displayed as a function of VehiclePerformance. Each datapoint is labeled with the assignment of technicians in the form (Technician1, Technician2, Technician3).

As before, we consider vehicle performance to be measured in terms of the amount of time greater than 40 hours in which all vehicles are operational. In this case however, we also add an addition performance function to account for the monetary cost required to maintain the fleet of vehicles. There are two sources of cost that we choose to model for this system. The first is the cost of a repair event. We associate a unique cost with the repair event for each component in the vehicles. No cost is associated with repair events for entire subsystems, as the individual components account for the cost required to bring the subsystem back online. The specific costs associated with each component are shown in Table 2.

The second contributor to the Cost performance function is the wages paid to the technicians. This cost is measured in terms of how much money is spent per hour retaining the technician to work on the vehicle fleet. Technicians are also assigned other billable jobs as well, and therefore the hourly cost for each technician with respect to this fleet of vehicles is only a portion of the entire wage for the technician. The cost per hour to retain Technician 1 and 2 is 2.0 respectively, while the cost to maintain Technician 3 is 3.25. The discrepancy between wages is due to the difference is skill levels.

We now have two performance functions: VehiclePerformance, which is identical to the performance function used in section 6.1, and Cost, which is defined in terms of repair costs and technician wages. Here again we are faced with a multi-objective optimization problem, where the goal is to maximize vehicle performance while minimizing cost. The controllable variables are the use of technicians. Formally we wish to identify a trajectory for the three technician nodes that will optimize the expected values for the performance functions. We start by constraining the search space. In our system, we either hire a technician or not, meaning that the state of the technician does not change throughout the entire time of interest. This reduces the problem to that of identifying the initial states for each technician node.

To find the optimal assignment of technicians, we set evidence that each technician node is either zero or one for the entire time period and evaluate the VehiclePerformance and Cost performance functions. We do this for every assignment of technician nodes, resulting in $2^3 = 8$ datapoints. Fig. 11 shows each assignment's cost plotted against the vehicle's performance. The datapoints are labeled as (t_1, t_2, t_3), where

t_n is the state assignment for Technician n. The goal is to minimize cost and maximize vehicle performance, so here again the best solution can be found in the lower right side of the graph. The Pareto frontier consists of the darker points, which dominate the remaining lighter colored nodes. Depending on the importance placed on the vehicle's performance vs cost, any of the assignments in the Pareto frontier are a valid choice for the optimal solution.

This experiment demonstrates how CTBNs can be used to solve complex optimization problems. Intuitively it may seem that as more technicians are employed, vehicle performance will increase along with cost. While this is true for vehicle performance, we see that the assignment with the minimal cost is not $(0, 0, 0)$, but rather $(0, 0, 1)$, which corresponds to assignment of Technician 3. This occurs because Technician 3 ultimately reduces the number of necessary repair events, each of which has a cost associated with it. The amount of money saved in repair events is greater than the amount spent on wages, and in the end Technician 3 actually saves money in the model. At first glance, it may seem tempting to spend less money on a Level I technician, but this model shows that there is absolutely no benefit to either cost or vehicle performance in hiring a Level I technician if a Level II technician is available. If vehicle performance is absolutely critical and money is less of a concern, hiring a Level I technicians to supplement a Level II technician can provide some increased performance. The interactions between technician skill, wages, vehicle performance, and repair cost are complex and not immediately obvious. Through the use of CTBNs, we are able to solve this multi-objective optimization problem which allows for more informed decision making.

7. 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. Finally we describe how the CTBN framework can be used to perform continuous-time optimization. We define the optimization process formally in terms of performance functions and a complete trajectory over a subset of nodes. To compensate for the computational complexity of the optimization procedure, we suggest the use of constraints that restricts the search space.

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. Furthermore we demonstrate the use of CTBNs for optimization using a family of performance functions on two separate networks. In the drug network example, we make use of constraints that restrict when transitions occur within the control variable. In the vehicle fleet example, we used a more restrictive constraint that no transitions occur, and optimize over the starting states for three separate variables. In both cases, the CTBN was able to find the Pareto frontier when optimizing over two conflicting performance functions. The optimization procedure enables the user to make an informed decision regarding the optimal choice in a complex dynamic system.

As future work, we intend to investigate automated methods for constraining the search space over the trajectory of the control variables. The experiments reported in this work made use of domain knowledge to reduce the set of necessary evaluations to a tractable size. We intend to evaluate the effectiveness of gradient based methods, as well as population-based optimization methods like particle swarm optimization. This would work by performing a directed search over the available search space, rather than evaluating every possible assignment in the set.

References

[1] D. Cao, Novel Models and Algorithms for Systems Reliability Modeling and Optimization, Ph.D. thesis, Wayne State University, 2011.

[2] I. Cohn, T. El-Hay, N. Friedman, R. Kupferman, Mean field variational approximation for continuous-time Bayesian networks, in: Proceedings of the Twenty-Fifth Conference Annual Conference on Uncertainty in Artificial Intelligence, UAI, AUAI Press, Corvallis, Oregon, 2009, pp. 91–100.

[3] M. Druzdzel, M. Henrion, Efficient reasoning in qualitative probabilistic networks, in: Proceedings of the Eleventh National Conference on Artificial Intelligence, AAAI, 1993, pp. 548–553.

[4] T. El-Hay, N. Friedman, R. Kupferman, Gibbs sampling in factorized continuous-time Markov processes, in: Proceedings of the Twenty-Fourth Conference Annual Conference on Uncertainty in Artificial Intelligence, UAI, AUAI Press, Corvallis, Oregon, 2008, pp. 169–178.

[5] Y. Fan, C. Shelton, Learning continuous-time social network dynamics, in: Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI, 2009, pp. 161–168.

[6] Y. Fan, J. Xu, C.R. Shelton, Importance sampling for continuous time Bayesian networks, J. Mach. Learn. Res. 99 (2010) 2115–2140.

[7] E. Gatti, Graphical Models for Continuous Time Inference and Decision Making, Ph.D. thesis, Università degli Studi di Milano-Bicocca, 2011.

[8] E. Gatti, D. Luciani, F. Stella, A continuous time Bayesian network model for cardiogenic heart failure, Flex. Serv. Manuf. J. (2011) 1–20.

[9] R. Herbrich, T. Graepel, B. Murphy, Structure from failure, in: Proceedings of the 2nd USENIX Workshop on Tackling Computer Systems Problems with Machine Learning Techniques, USENIX Association, 2007, pp. 1–6.

[10] R.T. Marler, J.S. Arora, Survey of multi-objective optimization methods for engineering, Struct. Multidiscip. Optim. 26 (6) (2004) 369–395.

[11] K.P. Murphy, Dynamic Bayesian Networks: Representation, Inference and Learning, Ph.D. thesis, University of California, 2002.

[12] B. Ng, A. Pfeffer, R. Dearden, Continuous time particle filtering, in: International Joint Conference on Artificial Intelligence, vol. 19, 2005, p. 1360.

[13] U. Nodelman, E. Horvitz, Continuous Time Bayesian Networks for Inferring Users' Presence and Activities with Extensions for Modeling and Evaluation, Tech. Report MSR-TR-2003-97, Microsoft Research, 2003.

[14] U. Nodelman, C. Shelton, D. Koller, Continuous time Bayesian networks, in: Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence, UAI, 2002, pp. 378–387.

[15] U. Nodelman, D. Koller, C. Shelton, Expectation propagation for continuous time Bayesian networks, in: Proceedings of the Twenty-First Conference Annual Conference on Uncertainty in Artificial Intelligence, UAI, AUAI Press, Arlington, Virginia, 2005, pp. 431–440.

[16] S. Qiao, C. Tang, H. Jin, T. Long, S. Dai, Y. Ku, M. Chau, PutMode: prediction of uncertain trajectories in moving objects databases, Appl. Intell. 33 (3) (2010) 370–386.

[17] D. Shi, X. Tang, J. You, An intelligent system based on adaptive CTBN for uncertainty reasoning in sensor networks, Intell. Autom. Soft Comput. 16 (3) (2010) 337–351.

[18] L. Sturlaugson, J.W. Sheppard, Factored performance functions with structural representation in continuous time Bayesian networks, in: The Twenty-Seventh International FLAIRS Conference, 2014, pp. 512–517.

[19] L. Sturlaugson, J.W. Sheppard, Inference complexity in continuous time Bayesian networks, in: Uncertainty in Artificial Intelligence, 2014.

[20] L. Sturlaugson, J.W. Sheppard, Sensitivity analysis of continuous time Bayesian network reliability models, SIAM/ASA J. Uncertain. Quantificat. 3 (1) (2015) 346–369.

[21] J. Xu, A Continuous Time Bayesian Network Approach for Intrusion Detection, Ph.D. thesis, University of California, 2010.

[22] J. Xu, C. Shelton, Continuous time Bayesian networks for host level network intrusion detection, in: W. Daelemans, B. Goethals, K. Morik (Eds.), Machine Learning and Knowledge Discovery in Databases, in: Lect. Notes Comput. Sci., vol. 5212, Springer, Berlin/Heidelberg, 2008, pp. 613–627.

[23] J. Xu, C.R. Shelton, Intrusion detection using continuous time Bayesian networks, J. Artif. Intell. Res. 39 (1) (2010) 745–774.