Reactive Bayesian Network Computation using Feedback Control: An Empirical Study

Ole J. Mengshoel, Abe Ishihara, and Erik Reed
Carnegie Mellon University
Moffett Field, CA 94035
{ole.mengshoel, abe.ishihara, erik.reed}@sv.cmu.edu

Abstract

This paper investigates the challenge of integrating intelligent systems into varying computational platforms and application mixes while providing reactive (or soft real-time) response. We integrate Bayesian network computation with feedback control, thereby achieving our reactive objective. As a case study we investigate fault diagnosis using Bayesian networks. While we consider the likelihood weighting and junction tree propagation Bayesian network inference algorithms in some detail, we hypothesize that the techniques developed can be broadly applied to achieve reactive intelligent systems. In this paper's empirical study we demonstrate reactive fault diagnosis for an electrical power system.

1 INTRODUCTION

Substantial progress has been made over the last few decades in the areas of learning and reasoning using Bayesian networks (BNs) [29]. While most BN inference problems are computationally hard (NP-hard or worse) in the general case [6, 33, 28], efficient algorithms have been developed and demonstrated in a wide range of automated reasoning applications including model-based diagnosis [19, 24, 32, 31].

There has recently been an explosion in the number and types of computers available, capable of running implementations of sophisticated algorithms including BN-based algorithms. This is a result of the advancements in both hardware and software platforms now powering computers, which range from smart phones and tablet PCs to high-end gaming machines and high-performance computing (HPC) clusters.

The proliferation of computers of highly varying capability provides new opportunities for AI systems and also raises several interesting research questions [21]. Can BN-based algorithms be implemented, adapted, or "wrapped" such that they reactively (or in soft real-time) compute meaningful results across a myriad of commercially available computers? Can algorithms be optimized such that users may generally continue to operate their computers normally while also running AI systems–that is, continue to check email, chat, and partake in social networking?

To start answering these questions, this paper develops an architecture that integrates Bayesian network computation and feedback control. The architecture contains two feedback loops, a lower-level inner loop and a higher-level outer loop. What drives the inner loop is the difference between desired completion time (or setpoint, set to achieve reactivity) and actual completion time. The goal of the outer loop is to trade off between higher-level issues such as speed of inference, resource allocation, and accuracy.

While the architecture is general, we have for validation purposes used experimental data from an electrical power network located at the NASA Ames Research Center [30]. We demonstrate our novel approach in the area of fault diagnosis for this electrical power system, and investigate the BN inference algorithms of likelihood weighting [34], variable elimination [20, 9], junction tree propagation [18, 35], and belief propagation [29, 26]. Results for five different computers and three different operating systems are demonstrated. We show system identification results for both junction tree propagation and likelihood weighting, and demonstrate outer loop control by successfully changing the setpoint and BN inference algorithm employed.

By integrating control theoretic techniques and uncertainty processing using BNs, this research aims to: improve the reactivity and adaptability of uncertainty processing in different applications; improve the timeliness of computation under dynamic workloads

on varying computational platforms (smart phones, GPUs, multi-core CPUs, Hadoop clusters, ...); and reduce the effort and cost associated with developing and deploying BN-based software applications. Our goals are similar to those of anytime algorithms [8, 37, 3], which provide a way to trade off between computation time and solution quality. However, our feedback approach is more general in that it handles both anytime and non-anytime algorithms. In fact, we show experimentally how our approach enables us to gracefully switch, on-line, from a non-anytime algorithm (e.g., junction tree propagation) to an anytime algorithm (e.g., likelihood weighting) while achieving real-time response.

We are investigating control theory as it applies to software and our actuators are computational actuators rather than control surfaces on an aircraft or wheels on a robotic vehicle. For example, our actuators may change the number of processes being handled by a computer or the input parameters to C or Java programs that implement algorithms for BN computation. In contrast, the great majority of previous control theory research has, with several notable exceptions in computing and software [12, 2], focused on physical systems governed by Newtonian mechanics. In particular, we know of no similar work in the area of BN computation, including computation for diagnosis or prognosis, where control theory is applied.

The approach developed in this paper applies in situations where the computational load on a computer running AI (here, BN) applications is a key concern and where, simultaneously, we can prioritize and control the computational processes. Specifically, we partition computational processes into *high-criticality*, *medium-criticality*, and *low-criticality*; the high-critical processes are reactive. Our experiments are in the area of electrical power system diagnosis. However, the techniques are more widely applicable and we now outline a few areas where this approach may prove powerful. Examples of high-criticality tasks in which Bayesian networks (and similar probabilistic graphical models) can be used and for which one might want to provide a reactive response include medical monitoring, sensor fusion, speech recognition on a smartphone, gesture recognition, and video recognition. Examples of low-criticality tasks, which would be down-prioritized, suspended or killed by our approach (depending on circumstances) include virus scanning, backups, disk defragmentation, software updates, and volunteer computing (such as Seti@Home and Folding@Home).

What these applications have in common is that reactivity is important for some computational processes. However, reactivity is not important enough to warrant the use of a hard real-time operating system, which is often used in aerospace and other applications where safety is paramount. Our approach is tailored to soft real-time settings where low-criticality tasks can, if needed, be down-prioritized, suspended or terminated in order to provide reactive response for high-criticality tasks.

The rest of this paper is organized as follows. In Section 2 we provide background on Bayesian network computation, feedback control, and anytime algorithms. Section 3 discusses our architecture integrating feedback control and BN inference algorithms. In Section 4 we present the experimental setting and electrical power system data, while Section 5 discusses empirical results. We conclude and outline future research in Section 6.

2 PRELIMINARIES

2.1 BAYESIAN NETWORK INFERENCE

BNs represent multivariate probability distributions and are used for reasoning and learning under uncertainty [29]. Probability theory and graph theory form the basis of BNs: Roughly speaking, random variables are represented as nodes in a directed acyclic graph (DAG), while conditional dependencies are represented as graph edges. Each node is parameterized by a conditional probability density or distribution. A BN is a compact representation of a joint probability distribution if its graph is relatively sparse.

Formally, we let $\boldsymbol{X}$ be the BN nodes, $\boldsymbol{E} \subset \boldsymbol{X}$ the evidence nodes, and $\boldsymbol{e}$ the evidence. While the nodes $\boldsymbol{X}$ can represent either discrete and continuous random variables, we focus in this paper on the discrete case in which a node $X \in \boldsymbol{X}$ has a finite number of states $\Omega(X) = \{x_1, \ldots x_m\}$. A BN factorizes a joint distribution $\Pr(\boldsymbol{X})$, and allows for different probabilistic queries to be formulated and supported by efficient algorithms; they all assume that nodes in $\boldsymbol{E}$ are clamped to values $\boldsymbol{e}$. Considering the remaining nodes $\boldsymbol{R} = \boldsymbol{X} - \boldsymbol{E}$, the probabilistic queries are with respect to the posterior distribution $P(\boldsymbol{R} \mid \boldsymbol{e})$. Specifically, computation of most probable explanations (MPEs) amounts to finding an MPE over $\boldsymbol{R}$, or $\text{MPE}(\boldsymbol{e})$. Computation of marginals (or beliefs) amounts to inferring for one or more query nodes $\boldsymbol{Q} \subseteq \boldsymbol{R}$, where $Q \in \boldsymbol{Q}$, the posterior probabilities $\Pr(Q \mid \boldsymbol{e})$ which we denote $\text{BEL}(\boldsymbol{Q}, \boldsymbol{e})$. In diagnosis using BNs [19, 24, 32, 31], the terminology health nodes $\boldsymbol{H}$, where $\boldsymbol{Q} = \boldsymbol{H}$, is often used. By picking, for each $Q \in \boldsymbol{Q}$, a most likely state $q \in \Omega(Q)$, we obtain the most likely states $\text{MLS}(\boldsymbol{Q}, \boldsymbol{e})$ from $\text{BEL}(\boldsymbol{Q}, \boldsymbol{e})$. Computation of the maximum a posteriori probability (MAP) generalizes MPE computa-

tion and finds a most probable instantiation over nodes $\boldsymbol{Q} \subseteq \boldsymbol{R}$, MAP($\boldsymbol{Q}, \boldsymbol{e}$). MAP can be approximated using MPE and MLS; these two approximations are of interest because of the greater computational complexity of MAP [28] compared to MPE and BEL [6].

Different BN inference algorithms can be used to perform the above BEL, MPE, and MAP computations. We distinguish between exact and inexact algorithms. Exact algorithms for BEL and MPE computation include belief propagation in singly connected BNs [29], junction tree propagation [18, 35], conditioning [29, 14], variable elimination [20, 9], and arithmetic circuit evaluation [7, 4]. Inexact algorithms such as loopy belief propagation[1] [29, 26] and likelihood weighting [34] have been used to compute marginals; they have also been used to compute MPEs [16, 25] and MAPs [28]. In this paper we focus on computation of marginals; however the framework also applies to other queries including MPE and MAP.

2.2 ANYTIME REASONING

An anytime algorithm improves its solutions according to the computational resource allocated to it, and returns an (approximate) answer if interrupted [8]. Fundamentally, this iterative improvement algorithm framework provides a way to trade off between computation time and solution quality. Anytime algorithms are a useful tool for real-time system design, and there are also results on the composition of anytime algorithms [37]. Originally, the anytime approach was developed for single agents, recently it was extended to handle multiple agents [3].

Among BN inference algorithms, stochastic local search [16, 28, 22, 25], likelihood weighting [34], loopy belief propagation [29, 26], and conditioning [29, 14] can be considered anytime algorithms. However, many important and high-performing algorithms—including junction tree propagation [18, 35], variable elimination [20, 9], and arithmetic circuit evaluation [7, 4]—are unfortunately not anytime algorithms.

2.3 FEEDBACK CONTROL

Feedback control involves the manipulation of a system in which data, either measured or estimated, is transformed and utilized to modify the behavior of the system. This behavior is typically defined in terms of metrics such as stability, boundedness, response time, or over-shoot. Improving such metrics by means of feedback control is typically motivated by the desire to achieve some higher level objective such as ensuring a comfortable ride on a passenger transport jet, an average production rate on a robotic assembly line, or a maximal duration of a credit card transaction.

Recently, control theory has been applied to computing systems including control of HTTP servers [11], email servers [27], quality of service assurance [36], internet traffic control [13], and load balancing. Typically, computing systems are different from traditional feedback control applications in robotics and aircraft. First, modeling of the plant does not typically start from Newtonian mechanics; rather it often begins with a black box approach. Second, actuation can in some cases be almost instantaneous, such as flipping a bit, or writing a short integer to memory, i.e. specifying the maximum number of connections to a server. Third, measurements are often non-noisy but delayed. In analog sensing, filtering is used to remove noise, and at the same time can introduce significant (and unwanted) phase lag. However, in computing systems, a more difficult delay appears at the measurement. In applications such as control of an email server, the (discrete) delay is associated with the completion of a Remote Procedure Call (RPC). This delay is usually not known. Finally, disturbances can significantly impact performance. Take the example of the IBM Domino server [12]: Certain combinations of requests, made independently by different users impact CPU utilization in a nonlinear manner; this can be regarded as a stochastic disturbance. These disturbances, which may depend on time of day and day of week, can have a significant impact.

3 CONTROLLING BAYESIAN NETWORK COMPUTATION

Our goal is to support the application of both non-anytime and anytime BN inference algorithms in reactive settings by introducing techniques from feedback control. A key concept in feedback control is the *plant*. Here, the plant to be controlled is a dynamic system operating in the discrete time domain. This definition encompasses computers including mobile phones, tablets, laptops, desktops, and multi-core systems. In this paper, the plant is a digital computer performing a high-criticality (or *reactive*) process. The reactive process runs, in our case, a BN used for diagnosis. We also assume that on this computer, low- and medium-criticality (or *background*) processes are running and are competing for CPU cycles, memory, and other computer resources. The impact that these background processes have on the reactive process and its ultimate outcome (detection of a failure event) depends on a number of parameters such as operating

[1] Pearl's main emphasis was originally on exact propagation in singly connected BNs, however he also mentioned inexact propagation in arbitrary BNs [29, Exercise 4.7],

system, RAM, processor type and clock-speed, CPU cache, etc. Since hardware and software vary immensely, this represents a source of significant uncertainty and presents challenges for both modeling and control.

We introduce the control system framework illustrated in Figure 1. The framework supports both anytime (and inexact) as well as non-anytime (and typically exact) algorithms, and distinguishes between inner and outer control loops because their objectives differ.

Inner Loop: The goal of the inner loop, which is traditional to feedback control, is to make careful adjustments according to the parameters set by the outer loop. These are key parameters in our integrated approach: $r(k)$ is desired completion time (or *Setpoint*) for sample time k. We would like the computational process to finish within this time. $y(k)$ is the actual completion time (or just *Actual*) for sample time k. $e(k) = r(k)$ - $y(k)$ is the error signal at time k; $u(k)$ is the maximal number of low-criticality processes (or *Max Processes*). $u(k)$, also known as the control law, is taken to be a proportional-derivative controller [12].

We partition computational processes into *high-criticality* (and having an $r(k)$ value associated with them), *medium-criticality*, and *low-criticality* (subject to actuation by control system). High-criticality processes are reactive and essential. Medium-criticality processes are non-reactive but essential. Low-criticality processes are non-reactive and non-essential. The actual (or measured) number of low-criticality processes (or *Actual Processes*) at sample time k is denoted $v(k)$.

Our approach uses simple black-box prediction techniques, as detailed below. It is the combination of this black-box approach with the use of a desired computation time (setpoint $r(k)$ in Figure 1) and an actual computation time (output $y(k)$ in Figure 1) that makes it work. The actual computation time is just a measurement of how long a BN computation took. The desired computation time depends on the frequency of our reactive process and other factors. For example, a frequency of 10 Hz means that the desired computation time is upper-bounded by 100 milliseconds.

Outer Loop: The goal of the outer loop is to jointly optimize speed of inference, resource allocation, accuracy, and other factors. Unlike inexact anytime algorithms, which have a similar objective, our approach can use exact algorithms. If, for example, the exact junction tree propagation and variable elimination algorithms are employed, reduced accuracy is not an issue. However, accuracy is in general important, and accuracy versus computation time trade-offs are performed in the outer loop of the framework. For inexact

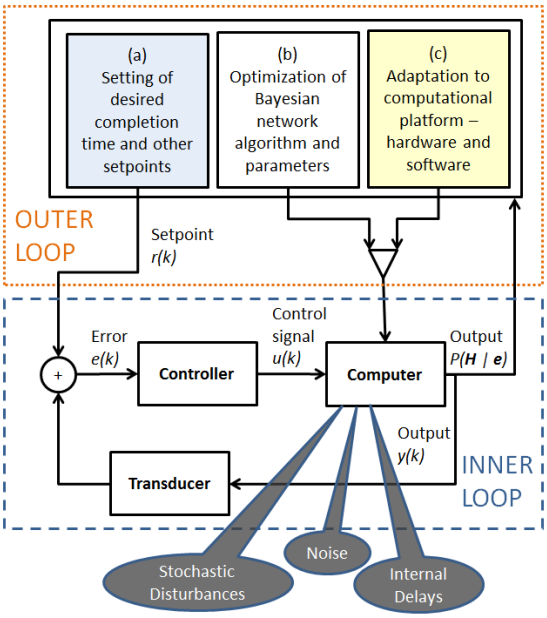


Figure 1: Our integrated architecture, where we wrap Bayesian network computation into two feedback control loops, a traditional inner loop where the Controller controls a Plant (here, a Computer) and a higher-level outer loop. While there are several options, this paper uses desired and actual computation time for setpoint $r(k)$ and output $y(k)$ respectively.

particle-based algorithms such as likelihood weighting and particle filtering, $p(k)$—the number of particles—is an example of such a parameter. One would like to use a very large number of particles for simulation, since this improves the accuracy of the estimate of the posterior $P(\boldsymbol{H}(k) \mid \boldsymbol{e}(k))$, however this may take too much time and one needs to carefully restrict the number of particles. In most experimental results with likelihood weighting reported in this paper, we assume that $p(k) = p$ and $r(k) = r$ are fixed (or stationary).[2]

4 METHODS AND DATA

System health management, which may utilize Bayesian network, can integrate information from heterogenous sensors and perform computation for the purpose of fault diagnosis, prognosis, and mitigation [19, 32, 31, 5]. Electrical power system fault diagnosis using Bayesian networks is the main focus of the experiments in this paper.

Data and Bayesian Network: We used experimental data from a real-world electrical power network, known as ADAPT, located at the NASA Ames Re-

[2] However, in Section 5.3 (see Figure 6) we discuss how and why $r(k)$ can be changed over time.

Computer	OS	Data Set	R^2	MSE
Laptop	Win7	1	0.824	0.606
Laptop	Win7	2	0.906	1.91
Desktop	Win7	1	0.810	0.315
Desktop	Win7	2	0.906	0.220
High End	Suse11	1	0.778	0.024
High End	Suse11	2	0.807	0.0051
Server	Ubu9	1	0.935	1.17
Server	Ubu9	2	0.946	0.450
Old Server	Ubu11	1	0.857	0.067
Old Server	Ubu11	2	0.813	0.0188

Table 1: Estimation of model parameters for different computers, operating systems (OSs), and data sets.

search Center [30]. ADAPT contains capabilities for power generation, power distribution, and loads representative for what can be found in aerospace vehicles. For the purpose of this paper we focus on a small part of ADAPT containing the following components: EY183 (relay), DC482 (DC load), E181 (voltage sensor), IT181 (current sensor), and ESH183 (relay feedback sensor). Scenarios are taken from Tier 2 of the DX 2009 competition data set.[3] These scenarios consist of nominal runs, with no faults in ADAPT, as well as runs involving one or more faults in components or sensors, diagnosed using BNT's implementation[4] of likelihood weighting (LW) [34], variable elimination (VE) [20, 9], junction tree propagation (JTP) [18, 35], loopy belief propagation (LBP) [26], and Pearl's belief propagation (PBP) [29].[5] It has previously been established that BNs perform very well in this domain, with the BN-based ProDiagnose system [32, 31] having the best performance in 3 of 4 of the DX international diagnostic challenges in 2009 and 2010.[6]

Computational Platforms: Five different computers, representing a broad spectrum of computing capability, were used. The *Laptop* computer is a 2.40GHz Intel i5 M520 with 4 cores, 8 GB of RAM, and 3 MB cache memory. The *Desktop* computer is a 3.20GHz AMD Phenom II X6 1090T with 6 cores, 8 GB of RAM, and 3 MB cache memory. The *High End* computer is a 2.00GHz Intel Xeon X7550 with 64 cores, 126 GB of RAM, and 18 MB cache memory. The *Server* computer is a 2.50GHz Intel Core 2 Quad Q8300 with 4 cores, 8GB GB of RAM, and 2 MB cache memory. The *Old Server* computer is a 2.80GHz Intel Xeon with 4 cores, 4 GB of RAM, and 1 MB cache memory. Broadly speaking, the High End computer is most powerful, due to its many cores and large memories, while the Old Server is least powerful due to its comparatively small memories. As reflected in Table 1, the operating systems Windows 7 (Win7) and several Linux variants–Ubuntu (Ubu9 and Ubu11) and Suse (Suse11) were employed in experiments.

Disturbance Generation: In the following, we model for simplicity low-criticality process disturbances to a computer (including user disturbances) as a Poisson process with rate λ. In the experiments, the low-criticality OS processes are CPU-intensive and execute mathematical operations in a tight loop; there is no I/O. This introduces a stochastic delay in the actuation of the plant: even if the control input $u(k)$ increases at the next time step, $u(k+1) > u(k)$, the probability of the number of processes actually increasing is low. The Poisson disturbance term is regarded as unmodeled dynamics and is not explicitly compensated for in the error term of the ARX model (1). It is of interest to model this phenomenon by a stochastic delay in the control input to the plant.

Note that our approach handles situations in which the resource consumption of low-criticality processes varies dramatically. For example, there can be many low-criticality processes (executing in parallel) that are mostly idle. On the other hand, even a few resource-hungry low-criticality processes could be too much to handle. If many "mostly idle" processes are present, our controller automatically sets the "Max number of processes" parameter $u(k)$ higher than if there are a few resource-hungry low-criticality processes. See Figure 6 and Figure 5 for how $u(k)$ is varied by our controller, due to the varying computational load of low-criticality and Bayesian network processes. The controller maps, see Figure 1, from a setpoint $r(k)$ (Computation time) to a control signal $u(k)$ (Max number of processes), and so is able to handle the varying resource use of different processes.

5 EMPIRICAL RESULTS

In this section we present experimental results on ADAPT data for our reactive approach, as enabled by feedback control and summarized in Figure 1.

5.1 COMPUTATION AS A PLANT

Control Input: The input to the plant (i.e., the computer in Figure 1), which ultimately is computed by a control algorithm, is denoted by $u(k)$. This input determines the number of low-criticality processes that are allowed to run at any given sample time. Denote the actual number of low-criticality processes by $v(k)$ where k denotes a sampling index. In the exam-

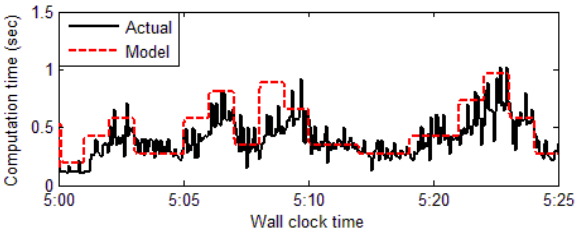
[3] http://www.dx-competition.org/

[4] http://code.google.com/p/bnt/

[5] The specific BNT functions we use are pearl_inf_engine (PBP), belprop_inf_engine (LBP), likelihood_weighting_inf_engine (LW), jtree_inf_engine (JT), and var_elim_inf_engine (VE).

[6] http://www.dx-competition.org/

(a) Junction tree propagation (JTP)



(b) Likelihood weighting (LW)

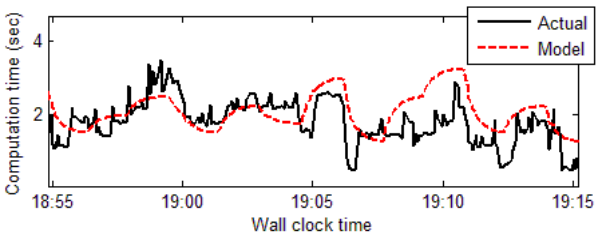


Figure 2: Results of system identification. Two different square waves used as input $u(k)$ to system identification, where we compare Actual completion time $y(k)$ and Model completion time $y_m(k)$ on a Laptop for (a) junction tree propagation versus (b) likelihood weighting.

ples that are used in this paper, if $v(k) \geq u(k)$, then processes are terminated until $v(k) < u(k)$.[7] The termination process is nearly instantaneous with respect to the sampling period.

Plant Modeling: There are several approaches to the modeling of computation for control applications. The approach taken here is termed linear Auto-Regressive modeling with eXogenous input (linear ARX). Nonlinear approaches, such as discrete time neural networks, may also be used. Nonlinear modeling is more complex yet may be able to capture inherent nonlinear behavior otherwise unaccounted for in ARX modeling. On the other hand, linear modeling is generally simpler to understand and implement.[8]

Suppose the plant is described by an ARX model with an additive noise term. Denote $P^{(i)}$, $y^{(i)}(k)$, $u(k)$, and $\eta(k)$, the plant operator, scalar output, input, and noise at sample time k, respectively. The integer i denotes a specific plant or device, such as a particular laptop or desktop. In general, we will assume that we have a finite class of plants $P^{(i)}$ for $i \in [1, M]$.

In general, the relationship between these quantities is given by:

$$y^{(i)}(k) = P^{(i)}u(k) = \phi^T(k-d)\theta^{(i)} + \eta(k) \qquad (1)$$

where $\phi^T(k-d)$ denotes the regression vector and consists of a tapped delay line of input and output measurements, and $\theta^{(i)}$ denotes a vector of real-valued parameters corresponding to the ith plant. A number of methods exist to estimate $\theta^{(i)}$ in both batch and online modes. Similar ARX models have been used in modeling a number of digital processes [12].

Open-loop Modeling and Generalization: Figure 2 shows the result of *open-loop* system identification, specifically the performance of a model where the parameters were estimated using least-squares (batch-mode). This figure compares, for varying sample time k, the *Actual* computation time $y(k)$ (see Figure 1) with our model's predicted computation time $y_m(k)$ (or just *Model*). For both JTP and LW, the rate of Poisson process creation is 45 seconds for LW and the sampling rate is $\frac{1}{5}$ Hz. Figure 2(a) depicts the performance for JTP, while Figure 2(b) shows LW. Overall, the fit between the model and actual behavior is quite good in both cases, perhaps slightly less so for JTP. The following may be the explanation why the JTP graph is more "blocky" and less of a fit than the LW graph: JTP is significantly faster than LW, thus JTP is more sensitive to the other computational processes.

Further details on how open-loop system identification is performed and generalizes are provided in Figure 3.[9] Here, for (1) we simplified $y^{(i)}(k)$ to $y(k)$ and used a first order discrete time model

$$y(k) = c_1 y(k-1) + c_2 u(k-1) + c_3 \qquad (2)$$

to model a specific plant's input output behavior. Input signals used to generate the training and testing data are shown in Figure 3(a), and consist of pseudo-random $u(k)$ square waves. Figure 3(b) shows the result of open-loop system identification for two different computers. Figure 3(b) shows the performance of the model where the parameters were estimated using least-squares (batch-mode), and depicts the performance of the laptop (top) and the server (bottom) using the regression parameters obtained via laptop-generated data. It is clear from these empirical results

[7] Currently, we randomly terminate one of the low-criticality OS processes. Additional information, such as process priority, can easily be used to guide termination.

[8] The ARX model was chosen for several reasons; most importantly it gave solid system identification performance. Allowing the plant model itself to be a probabilistic graphical model would be interesting, and stochastic approaches (based on stochastic differential equations) have been developed in control theory [10, 15]. In light of its performance, we believe that the standard ARX approach is well-justified and leave stochastic control to future work.

[9] While not illustrated in Figure 3, we note that closed loop system identification is also required and presents additional data challenges, for example, data collinearities.

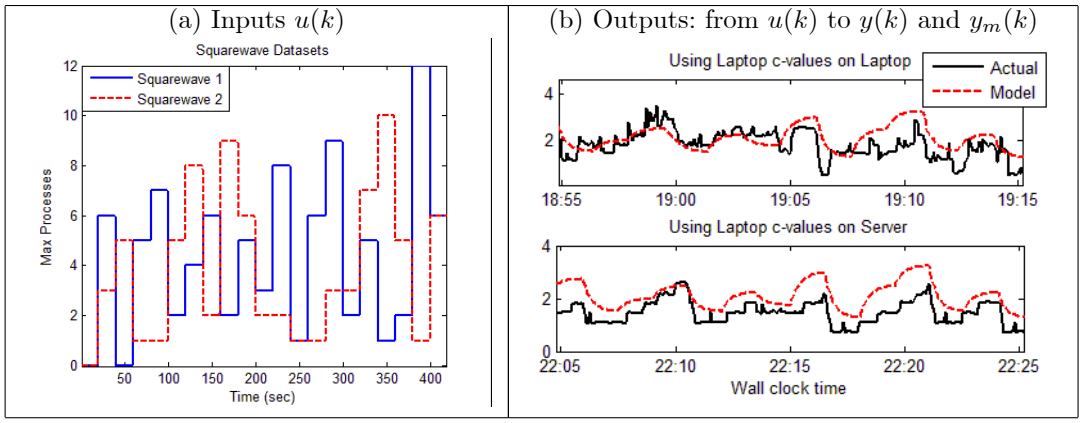


Figure 3: System identification and generalization. (a) Two different squarewaves, Squarewave 1 and Squarewave 2, used as input $u(k)$. Squarewave 1 was used for training of $y_m(k)$ and Squarewave 2 was used for testing. (b) Application to two different computers, comparing Actual completion time $y(k)$ and Model completion time $y_m(k)$: (top) Laptop–good fit and (bottom) Server–decent fit.

that models obtained on one platform (here, laptop) performs best on that platform. The models are likely to perform less well on another platform (here, server), but are still of some value.

5.2 INNER LOOP CONTROL

The objective of the control system is to minimize the error signal given by $e(k) = r(k) - y(k)$ where $r(k)$ denotes the reference or desired computational time required for BN computation. The output, $y(k)$, denotes the actual time the computer takes to complete the BN computation and generate the posterior probabilities BEL($\boldsymbol{Q}, \boldsymbol{e}$) as discussed above.

From the control engineering perspective, there are several key issues: (1) *Static uncertainty in plant parameters*: A tablet PC running Windows Vista will behave differently from a High Performance Computing (HPC) cluster running openSUSE Linux. A control system designed for one computer may not perform well on another. In other words, the control system parameters are uncertain and need to be estimated. (2) *Stochastic disturbances*: The process that determines when a low-criticality process is generated by a user is, in general, stochastic. This is, in itself, an active area of research [1]. Furthermore, the impact a particular low-criticality process has on the high-criticality process may vary substantially. (3) *Stochastic delay in the input process*: When the control input $u(k)$ increases, the actual number of processes may or may not increase. This represents a stochastic delay in the actuation of the digital system being controlled.

To illustrate how our approach handles the above, we optimized a linear controller given by the following Z-transfer function: $H(z) = \frac{K}{1-\alpha z^{-1}}$, where K and α are real-valued controller gain parameters. $H(z)$ is the transfer function from the error signal $e(k)$ to the input to the plant $u(k)$. Performance for a fixed setpoint is shown in Figure 6(a). Computation time is maintained at approximately $r(k) = 2$ sec, however since this is a soft real-time approach there are excursions above this setpoint.

5.3 OUTER LOOP CONTROL

Our proposed research is based on controlling BN computation in an inner loop as well as in an outer loop (see Figure 1); so far we discussed the inner loop. We now briefly discuss the outer loop, where the output BEL($\boldsymbol{Q}, \boldsymbol{e}$) of BN computation, as well as other factors external to the inner loop, may change desired completion time and also plant behavior (sampling frequency, BN computation algorithm, number of particles assuming a simulation algorithm, etc.).

Adaptation to Computational Platform (see Figure 1(c)): Table 1 summarizes results of running likelihood weighting on the ADAPT BN, using five different computers and three different operating systems. System identification was performed on all computers using the two input data sets shown in Figure 3(a). Batch least squares was used to determine the parameters of a first order linear ARX model. Table 1 shows the resulting R^2 and mean squared error (MSE) of the parameter fit for the various computers and operating systems. In most cases, we observed a low MSE.[10]

[10]It is important to note the underlying variations in the ARX model parameters, which are used by the control design process. The diverse set of parameters, omitted here to save space, illustrate the benefit of using techniques from adaptive feedback control, specifically learning from

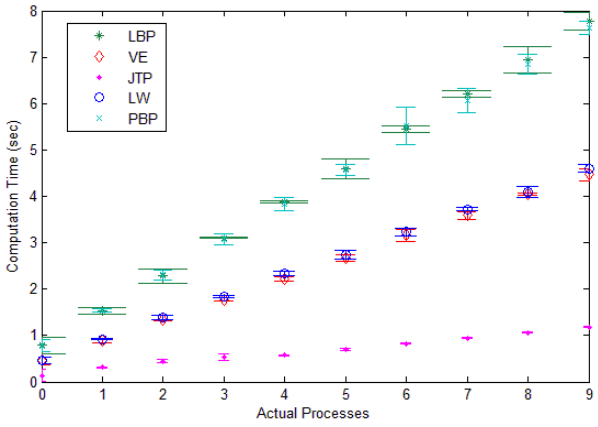


Figure 4: Actual computation time $y(k)$ as a function of actual number of processes $v(k)$ for five different BN inference algorithms LBP, VE, JTP, LW, and PBP running on High End computer.

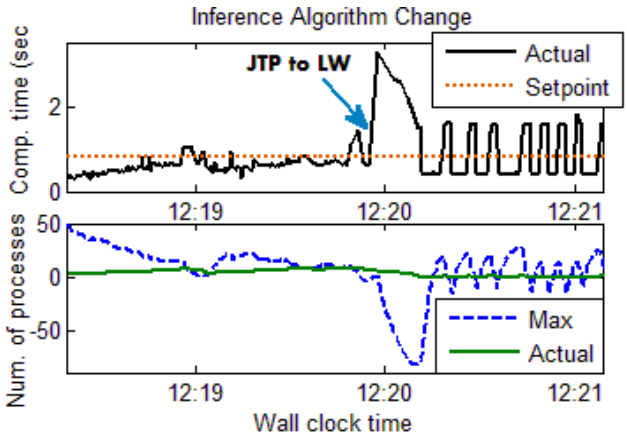


Figure 5: Experiment on Laptop using 1 Hz sampling rate, showing a successful switch of BN inference algorithm from junction tree propagation (JTP) to likelihood weighting (LW) using feedback control.

Optimizing BN Algorithm and Parameters (see Figure 1(b)): In Figure 4, we show the steady state computation time results for the five BN inference algorithms discussed above. Here, steady state is defined loosely as the time k^* such that $y(k+1) = y(k) = y_{ss}$ for all $k \geq k^*$. Plugging this into (2) yields $y_{ss} = \frac{c_2 u_{ss} + c_3}{1 - c_1}$, where $u(k) = u_{ss} \ \forall k \geq k^*$. Thus, the linear gain coefficient (slopes in Figure 4) for each algorithm is given by $\frac{c_2}{1-c_1}$.

The results fall into three groups: fast (JTP), medium (VE and LW), and slow (LBP and PBP) computations. Also, the standard error in computation time varies between the algorithms, with JTP again being the best with a very small standard error.

While JTP is exact and fast, its Achilles' heel is memory consumption [23]. Consequently, it can be necessary, when running other memory-intensive processes, to use a less memory-intensive but inexact algorithm like LW. How should one switch between two algorithms, say the non-anytime algorithm JTP and the anytime algorithm LW, that have very different computational resource requirements but operate on the same BN? Feedback control can help in this regard, see Figure 5. From a control perspective, this is considered *dynamic uncertainty in plant parameters*: Given a computer $P^{(i)}$, a change in the BN algorithm will impact the way the plant, $P^{(i)}$, responds. Here, we switch from JTP to LW around 12:20, while maintaining the setpoint (on average) after a transient period lasting around 10 seconds.[11] This is, to the best of our knowledge, the first demonstration of a successful on-line switch between two very different inference algorithms (from JTP to LW) while a desired completion time is maintained.

Changing the Setpoint (see Figure 1(a)): There is both a supply side and a demand side in computing. On the supply side, one can control the supply of computational resources, in the form of computers, CPUs, CPU threads, or GPU threads. On the demand side, the outer loop can vary the Setpoint $r(k)$, perhaps in combination with varying other outer loop parameters such as sampling frequency $f_C(k)$ and number of particles $p(k)$ used in likelihood weighting [34] or particle filtering [17].

One reason for the outer loop to vary the computational resources allocated to the high-criticality process is illustrated in the following. Suppose, for $k < k^*$, that $P(\boldsymbol{H}(k) \mid \boldsymbol{e}(k))$ suggested that there was one or more faults in the ADAPT electrical power system, while $P(\boldsymbol{H}(k^*) \mid \boldsymbol{e}(k^*))$ indicated that this was a false alarm. In this case, we may want to be less stringent about ensuring that computation of $P(\boldsymbol{H}(k^*+1) \mid \boldsymbol{e}(k^*+1))$, $P(\boldsymbol{H}(k^*+2) \mid \boldsymbol{e}(k^*+2))$, etc. finishes in a timely fashion, in other words it makes sense to put $r(k^*+1) > r(k^*)$. Figure 6(b) shows how this type of step change, at $k^* \approx 23{:}28$, is supported by our control-theoretic framework. The baseline of not varying $r(k)$ is shown in Figure 6(a). We here assume that sample frequency $f_C(k) = f_C$ is constant, and consider (i) $r(k-1) = r(k) < 1/f_C$ (as when $r(k) = 2$ sec in both Figure 6(a) and Figure 6(b)) versus (ii) $r(k) \approx 1/f_C$ (as when $r(k) = 4$ sec Figure 6(b)). The advantage of (i) is that there is a much greater chance that BN computation finishes before a new computational cy-

the computational environment and adapt to changes.

[11] Note, our approach does not make any *hard* real-time guarantees, only *soft* ones, and consequently the actual computation time is sometimes greater than the setpoint.

(a) Fixed Setpoint $r(k)$ (b) Varying Setpoint $r(k)$

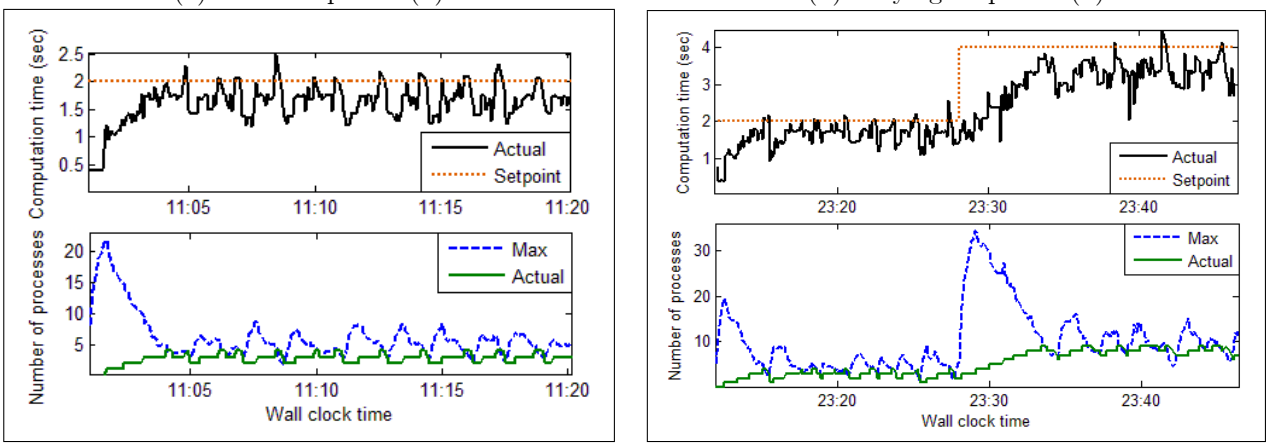


Figure 6: Outerloop optimization for LW, using: (a) fixed Setpoint $r(k) = 2$ and (b) change of Setpoint, approximately at time 23:28, from $r(k) = 2$ to $r(k) = 4$.

cle starts, which is essential when EPS faults are more likely. The advantage of (ii), on the other hand, is that more low-criticality processes are allowed to run.

The trade-off between fast inference for the high-criticality process versus running many low-criticality processes is illustrated in Figure 6. Figure 6(b)'s bottom panel shows an increase in the number of processes, on average, as a result of the increase in $r(k)$ at $k^* \approx$ 23:28; no similar increase can be found in Figure 6(a)'s bottom panel.

6 DISCUSSION AND OUTLOOK

Deploying BN algorithms and other resource-intensive AI algorithms can be a challenge when there are non-trivial constraints on computational resources in applications. In this paper, we have focused on supporting requirements for reactive response without requiring dedicated hard real-time computational resources according to worst-case computational requirements. We have focused on reactive diagnosis using BNs, motivated by domains with some but uncertain domain knowledge (hence probabilistic graphical models, specifically BNs) as well as uncertainty with respect to the computational platform and environment (hence feedback control).

An alternative approach to achieving reactive response is the use of anytime algorithms. The motivation behind anytime inference—namely the goal of intelligent and reactive systems—and our work is quite similar. However, the approaches are very different. Anytime algorithms are inherently inexact and produce solutions whose quality gradually improve with computation time [37]. We focus on what can done, on the computing system level, for a broad range of existing Bayesian network inference algorithms including both exact and inexact algorithms. As a consequence, our approach enables the use of exact but non-anytime algorithms (like variable elimination [20, 9], junction tree propagation [18, 35], and arithmetic circuit evaluation [7, 4]) in reactive settings. The benefit of this is that such exact algorithms often perform very well, however they do have limitations and consequently there are situations where they are unsuitable. With our approach, one can use these exact algorithms and then switch to an inexact (often anytime) algorithm only if needed, rather than having to always use an anytime algorithm.

We are in this paper using rather basic control theory ideas. This enables new results and many opportunities for future work, both theoretical and experimental, and we invite other researchers to participate in the exploration of this exciting area of reseach. We are, for example, developing adaptive control methods that leverage online system identification of the process. There are also many interesting research opportunities related to the use of BN posteriors as well as their accuracy, and perhaps multiple BNs at different levels of detail, in the outer control loop. In this area, there is a strong connection to anytime algorithms and metareasoning that can be further investigated, enabling more reactive and more capable intelligent systems.

Acknowledgements

This material is based upon work supported, in part, by NSF awards CCF0937044 and ECCS0931978.

References

[1] T. Beauvisage. Computer usage in daily life. In *Proc. of the 27th International Conference on Human factors in Computing Systems (CHI-09)*, pages 575–584, Boston, MA, 2009.

[2] Y. Brun, G. Di Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw. Engineering self-adaptive systems through feedback loops. In B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee, editors, *Software Engineering for Self-Adaptive Systems*, pages 48–70. Springer-Verlag, 2009.

[3] A. Carlin and S. Zilberstein. Decentralized monitoring of distributed anytime algorithms. In *Proc. of the Tenth International Conference on Autonomous Agents and Multiagent Systems*, pages 157–164, Taipei, Taiwan, 2011.

[4] M. Chavira and A. Darwiche. Compiling Bayesian networks using variable elimination. In *Proc. of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 2443–2449, Hyderabad, India, 2007.

[5] A. Choi, A. Darwiche, L. Zheng, and O. J. Mengshoel. A tutorial on Bayesian networks for system health management. In A. Srivastava and J. Han, editors, *Data Mining in Systems Health Management: Detection, Diagnostics, and Prognostics*. Chapman and Hall/CRC Press, 2011.

[6] F. G. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42:393–405, 1990.

[7] A. Darwiche. A differential approach to inference in Bayesian networks. *Journal of the ACM*, 50(3):280–305, 2003.

[8] T. Dean and M. S. Boddy. An analysis of time-dependent planning. In *Proc. of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, pages 49–54, St. Paul, MN, 1988.

[9] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.

[10] H. Deng, M. Krstic, and R. J. Williams. Stabilization of stochastic nonlinear systems driven by noise of unknown covariance. *IEEE Transactions on Automatic Control*, 46(8):1237 – 53, 2001.

[11] Y. Diao, N. Gandhi, J. L. Hellerstein, S. Parekh, and D. M. Tilbury. Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache Web server. In *Proc. of Network Operations and Management Symposium (NOMS-02)*, pages 219–234, 2002.

[12] J. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. Wiley, 2004.

[13] C. Hollot, V. Misra, D. Towsley, and W. Gong. On designing improved controllers for AQM routers supporting TCP flows. In *Proc. of IEEE INFOCOM*, pages 1726–1734, 2000.

[14] E. J. Horvitz, H. J. Suermondt, and G. F. Cooper. Bounded conditioning: Flexible inference for decisions under scarce resources. In *Proc. of the Fifth Conference on Uncertainty in Artificial Intelligence (UAI-89)*, pages 182–193, Windsor, Ontario, 1989.

[15] A. K. Ishihara, J. van Doornik, and S. Ben-Menahem. Stochastic stability of a neural-net robot controller subject to signal-dependent noise in the learning rule. *International Journal of Adaptive Control and Signal Processing*, 24(6):445–466, 2010.

[16] K. Kask and R. Dechter. Stochastic local search for Bayesian networks. In *Proc. of the Seventh International Workshop on Artificial Intelligence and Statistics (AISTATS-99)*, Fort Lauderdale, FL, Jan 1999.

[17] D. Koller and U. Lerner. Sampling in Factored Dynamic Systems. In A. Doucet, J. F. G. de Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods In Practice*, pages 445–464. Springer-Verlag, 2001.

[18] S. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems (with discussion). *Journal of the Royal Statistical Society series B*, 50(2):157–224, 1988.

[19] U. Lerner, R. Parr, D. Koller, and G. Biswas. Bayesian fault detection and diagnosis in dynamic systems. In *Proc. of the Seventeenth national Conference on Artificial Intelligence (AAAI-00)*, pages 531–537, 2000.

[20] Z. Li and B. D'Ambrosio. Efficient inference in Bayes nets as a combinatorial optimization problem. *International Journal of Approximate Reasoning*, 11(1):55–81, 1994.

[21] O. J. Mengshoel. Designing resource-bounded reasoners using Bayesian networks: System

health monitoring and diagnosis. In *Proceedings of the 18th International Workshop on Principles of Diagnosis (DX-07)*, pages 330–337, Nashville, TN, 2007.

[22] O. J. Mengshoel. Understanding the role of noise in stochastic local search: Analysis and experiments. *Artificial Intelligence*, 172(8-9):955–990, 2008.

[23] O. J. Mengshoel. Understanding the scalability of bayesian network inference using clique tree growth curves. *Artificial Intelligence*, 174:984–1006, 2010.

[24] O. J. Mengshoel, M. Chavira, K. Cascio, S. Poll, A. Darwiche, and S. Uckun. Probabilistic model-based diagnosis: An electrical power system case study. *IEEE Trans. on Systems, Man, and Cybernetics*, 40(5):874–885, 2010.

[25] O. J. Mengshoel, D. Roth, and D. C. Wilkins. Portfolios in stochastic local search: Efficiently computing most probable explanations in Bayesian networks. *Journal of Automated Reasoning*, 46(2):103–160, 2011.

[26] K. P. Murphy, Y. Weiss, and M. I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proc. of the Fifteenth Conference on Uncertainty in AI (UAI-99)*, pages 467–475, 1999.

[27] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. Using control theory to achieve service level objectives in performance management. *Real-Time Systems*, 23(1):841–854, 2002.

[28] J. D. Park and A. Darwiche. Complexity results and approximation strategies for MAP explanations. *Journal of Artificial Intelligence Research (JAIR)*, 21:101–133, 2004.

[29] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.

[30] S. Poll, A. Patterson-Hine, J. Camisa, D. Garcia, D. Hall, C. Lee, O. J. Mengshoel, C. Neukom, D. Nishikawa, J. Ossenfort, A. Sweet, S. Yentus, I. Roychoudhury, M. Daigle, G. Biswas, and X. Koutsoukos. Advanced diagnostics and prognostics testbed. In *Proc. of the 18th International Workshop on Principles of Diagnosis (DX-07)*, pages 178–185, Nashville, TN, 2007.

[31] B. W. Ricks, C. Harrison, and O. J. Mengshoel. Integrating probabilistic reasoning and statistical quality control techniques for fault diagnosis in hybrid domains. In *Proc. of the Annual Conference of the PHM Society 2011 (PHM-11)*, Montreal, Canada, 2011.

[32] B. W. Ricks and O. J. Mengshoel. Methods for probabilistic fault diagnosis: An electrical power system case study. In *Proc. of Annual Conference of the PHM Society, 2009 (PHM-09)*, San Diego, CA, 2009.

[33] D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82:273–302, 1996.

[34] R. Shachter and M. Peot. Simulation approaches to general probabilistic inference on belief networks. In *Uncertainty in Artificial Intelligence 5*, pages 221–231, Amsterdam, 1990. Elsevier.

[35] P. P. Shenoy. A valuation-based language for expert systems. *International Journal of Approximate Reasoning*, 5(3):383–411, 1989.

[36] C.-Z. Xu, B. Liu, and J. Wei. Model predictive feedback control for QoS assurance in webservers. *Computer*, 41:66–72, March 2008.

[37] S. Zilberstein and S. Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 82:181–213, 1996.