Adaptive Control of Bayesian Network Computation

Erik Reed
Carnegie Mellon University
NASA Research Park
Moffett Field, CA 94035
erikreed@cmu.edu

Abe Ishihara
Carnegie Mellon University
NASA Research Park
Moffett Field, CA 94035
abe.ishihara@sv.cmu.edu

Ole Mengshoel
Carnegie Mellon University
NASA Research Park
Moffett Field, CA 94035
ole.mengshoel@sv.cmu.edu

Abstract—**This paper considers the problem of providing, for computational processes, soft real-time (or reactive) response without the use of a hard real-time operating system. In particular, we focus on the problem of reactively computing fault diagnosis by means of different Bayesian network inference algorithms on non-real-time operating systems where low-criticality (background) process activity and system load is unpredictable.**

To address this problem, we take in this paper a reconfigurable adaptive control approach. Computation time is modeled using an ARX model where the input consists of the maximum number of background processes allowed to run at any given time. To ensure that the reactive (high-criticality) diagnosis is computed within a set time frame, we introduce a minimum degree pole placement controller to impose a limit on the maximum number of low-criticality processes. Experimentally, we perform electrical power system diagnosis using a Bayesian network model of and data from a NASA electrical power network. The Bayesian network inference algorithms likelihood weighting and junction tree propagation are successfully applied and changed mid-simulation to investigate how inference computation time changes in an unpredictable operating system, as well as how the controller reacts to inference algorithm changes.

I. Introduction

Probabilistic graphical models can be a valuable method for real time diagnosis of system health in electrical power systems and aerospace systems [1]. One class of probabilistic graphical models, Bayesian networks (BNs), are directed, acyclic graphs capable of concisely representing cause and effect using nodes and edges.

Each BN node represents a random variable that can be observed or inferred based on the behavior of other nodes via edges, which represent conditional probability distributions between nodes. For example, a BN model of an electrical system could represent each hardware component as a set of nodes, each with a set of discrete states, such as low, medium, and high; in the case of a node issuing a command, this could be on or off. Sensor components are observed nodes, whereas the state of a hidden component, like the health of a motor, can be inferred by using evidence in the system to compute a most likely explanation. Efficient BN algorithms have been developed to perform a wide range of automated reasoning, including model-based diagnosis [2].

Our research investigates the integration of control theory with AI algorithms, in particular BN inference, to allow computation to reactively operate in an operating system that is not a hard real-time operating system. We are interested in how model-based diagnoses interacts with an operating system (OS) on a device when performing inference. The device could potentially have CPU load due to active involvement by a user or due to other computational processes. Such processes are typically unpredictable but can be controlled. We distinguish between three types of processes running on the OS: high-criticality (reactive and essential), medium-criticality (non-reactive and essential), and low-criticality (non-reactive and non-essential). Which process falls into which of the tree types is application-dependent; in this paper the BN-based diagnosis process is high-criticality.

Our integration of control theory to software controls no physical system; actuators instead control input parameters to programs, the running of scripts and tasks, or the scheduling and process management of other programs running in the operating system. The majority of previous control theory research has dealt with physical systems with some notable exceptions [3], [4]. We know of no previous research in the area of BN computation for systems health diagnosis where control theory is applied.

This paper will proceed by introducing BN notation and related in work in Section II. Our controller, the method of controlling low-criticality processes, and process creation will be detailed in Section III. Experimental results for reactive BN control and modeling are discussed in Section IV. We summarize our results and outline future work in Section V.

II. Preliminaries and Related Work

A. Bayesian Network Inference

Let $\boldsymbol{X}$ be the BN nodes, $\boldsymbol{E} \subset \boldsymbol{X}$ the evidence nodes, and $\boldsymbol{e}$ the evidence. A BN factorizes a joint distribution $\Pr(\boldsymbol{X})$, and allows for different probabilistic queries to be formulated and supported by efficient algorithms as we will further discuss below; they all assume that all nodes in $\boldsymbol{E}$ are clamped to values $\boldsymbol{e}$. Computation of most probable explanations (MPEs) amounts to finding a most probable explanation over the remaining nodes $\boldsymbol{R} = \boldsymbol{X} - \boldsymbol{E}$, or MPE($\boldsymbol{e}$). Computation of marginals (or beliefs) amounts to inferring the posterior probabilities over one or more query nodes $\boldsymbol{Q} \subseteq \boldsymbol{R}$, specifically BEL($Q, \boldsymbol{e}$) where $Q \in \boldsymbol{Q}$. (In diagnosis using BNs, the terminology health nodes $\boldsymbol{H}$, where $\boldsymbol{Q} = \boldsymbol{H}$, is often used.) Marginals are used to compute most likely values (MLVs) simply by picking, in BEL($Q, \boldsymbol{e}$), a most likely state.

Different BN inference algorithms can be used to perform the above computations. We distinguish between exact and inexact algorithms. Exact algorithms include join tree propagation, conditioning, variable elimination, and arithmetic circuit evaluation. Inexact algorithms include likelihood weighting and stochastic local search. Both exact and inexact algorithms have been used to compute marginals and MPEs.

The motivation behind anytime inference [5] and our work is similar. Anytime inference algorithms are typically local search algorithms that have a solution at all times, allowing them to terminate at any point and return a solution estimate. However, the approaches taken are fundamentally different and we focus on what can done, on the computing system level, for existing Bayesian network inference algorithms including non-anytime algorithms. Our approach enables the use of high-performing but non-anytime algorithms (like variable elimination and junction tree propagation [6]) in reactive settings.

B. Feedback Control in Computing Systems

Recently, control theory has been applied to computing systems including control of HTTP servers [7], [8], email servers [9], quality of service assurance [10], internet traffic control [11], and load balancing. Computing systems have some characteristics that are typically not seen in traditional 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 removed noise, and at the same time can introduce significant (and unwanted) phase lag. However, in computing systems, a more difficult delay may appear 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 (uncertain). Finally, disturbances can significantly impact performance. Take the example of the IBM Domino server [3]: Clients make requests to the Lotus Notes server via RPCs. Assuming the request is processed, the impact on CPU usage varies significantly depending on the request. Sometimes, certain combinations of requests made independently by different users impact CPU utilization in a nonlinear manner; this can be regarded as a stochastic disturbance. However, from the point of view of the control objective, which is to regulate “RPCs in the server” (RIS), these disturbances can have a significant impact. These disturbances may depend on time of day and day of week.

C. Reconfigurable Control using Modified Recursive Least Squares

Controllers for computing systems must be designed to run on a variety of platforms (hardware and software) and account for unanticipated changes and/or failures that may occur on-line. As both hardware and software platforms advance at an enormous pace, it is desirable for a control system to be able to adapt and *keep up with* the myriad of possibilities that exist. Changes that impact the input-output dynamics include different operating systems, unanticipated low-criticality processes, and even different hardware platforms (iPhone versus a quad-core Linux workstation). A fixed controller, designed to optimize performance of a particular nominal or unfailed system, can result in degradation of performance and in some cases, loss of stability.

There are, in general, two control design approaches to tackle this problem, robustness and reconfigurability. A *robust* control design approach takes into account all possible variations in system parameters due to change or failure. The resulting control laws are insensitive to these varying parameters in the sense that the output does not vary significantly from the nominal performance criteria. The *reconfigurable* approach attempts to identify these changes in system parameters, and adjust the controller on-line, so that performance is not compromised due to an unforeseen failure.

We consider, in this paper, the reconfigurable approach. That is, we identify the changes in the parameters during plant changes and adapt the controller so that performance is maintained. This approach is termed *explicit* adaptive control, in which we must use an *explicit* parameter identifier. On the other hand, implicit adaptive control attempts to directly adjust the controller gains without the use of a separate parameter identifier.[1]

The underlying control approach taken is the Minimum Degree Pole Placement (MDPP) control design approach proposed by Astrom [12]. In this approach, feedfoward and feedback controllers are designed to force the open-loop system to follow a reference model. In the ideal case, model following is achieved perfectly. However, due to parametric uncertainty, this is not achieved in practice. This is the motivation to introduce reconfiguration to the control problem. During a change in plant dynamics the control effectiveness may change, requiring a larger or smaller actuator signal to maintain desired performance.

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 or 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.

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 set of plants $P^{(i)}$ for $i \in [1, M]$.

[1]Explicit adaptive control is also referred to as indirect adaptive control. Implicit adaptive control is also referred to as direct adaptive control.

The relationship between these quantities is given by:

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

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 parameters corresponding to the ith plant. A number of methods exist to estimate $\theta^{(i)}$ in both batch and on-line modes. Similar ARX models have been used in modeling a number of digital processes [3].

The reconfigurable control approach estimates θ on-line and then uses the estimates to update a control law (see Section III-B) below. To estimate θ, it is typical to use the recursive least squares (RLS) method, in which we minimize the cost function

$$J = \sum_{i=1}^{N} \lambda^{N-i} e^2(i), \tag{1}$$

where $e(i)$ is the error between the true and estimated outputs, and $\lambda \in (0,1)$ is the forgetting factor.

The RLS method can lead to two problems when attempting to track varying parameters. First, a small forgetting factor needed to track fast or abrupt parameter variations can cause a large covariance matrix which could lead to covariance "blow up." Second, as one decreases the forgetting factor, the size of the data window gets smaller and it is more likely that there will exist data collinearities within the data window. To deal with this issue, the Modified Recursive Least Squares (MRLS) algorithm prevents singularities in the covariance matrix by reformulation of the least squares problem [13]. The cost function to be minimized includes additional penalties on changes in the parameter values in the form of temporal and spatial constraints. Weighting matrices are included to adjust the extent of penalization on each parameter variation. For this case, we define the cost function as

$$J = \sum_{k=1}^{N} \|y(k) - x^T\theta(N)\|^2 \lambda^{N-k} + m\|\theta(N) - \theta(N-1)\|^2.$$

Here, $\theta(N)$ are the parameter estimates at time N.

Setting $\frac{dJ}{d\theta} = 0$, we get the solution,

$$\theta(N) = \left[\mathcal{X}^T\tilde{\mathcal{X}} + mI\right]^{-1}\left[m\theta(N-1) + \tilde{\mathcal{X}}^T y(N)\right],$$

where

$$\begin{aligned} \mathcal{X}^T &= \begin{bmatrix} x(1) & x(2) & \dots & x(N) \end{bmatrix} \\ \tilde{\mathcal{X}}^T &= \begin{bmatrix} x(1)\lambda^{N-1} & x(2)\lambda^{N-2} & \dots & x(N) \end{bmatrix} \\ y^T(N) &= \begin{bmatrix} y(1) & y(2) & \dots & y(N) \end{bmatrix} \end{aligned}$$

We note that the covariance matrix of this algorithm is now

$$P(t) = \left[\mathcal{X}^T\tilde{\mathcal{X}} + mI\right]^{-1}.$$

A recursive version of MRLS [14] is given as

$$\begin{aligned} \theta(t+1) &= \theta(t) + P(t+1)x(t+1)\left[y(t+1) - x^T(t+1)\theta(t)\right] \\ &\quad + m\lambda P(t+1)\left(\theta(t) - \theta(t-1)\right). \end{aligned}$$

To find $P(t+1)$, it is typical to use the Matrix Inversion Lemma (MIL). However, due to the additional penalty term, the MIL leads to the inversion of a $(1+m)\times(1+m)$ matrix, where m is the number of unknown parameters. Thus, we opt to directly take the inverse. For higher order systems, this approach may not be acceptable. Ways to reduce the order of this computation have been proposed [14].

Upon an abrupt plant failure, we would like to adapt our controller when the parameter estimates converge. The convergence time may vary depending on factors such as input excitation. We monitor the aposterior prediction errors,

$$\text{Prediction Error} = \left|y(t) - x^T(t)\theta(t-1)\right|. \tag{2}$$

We then define some Error Tolerance, and adapt our controller when

$$\text{Prediction Error} > \text{Error Tolerance}.$$

III. Framework and Methods

A. Bayesian Network Computation as a Plant

The plant to be controlled is a dynamic system operating in the discrete time domain. For our purposes, this plant definition encompasses computers including mobile phones, tablets, net-books, laptops, desktops, and high-end multi-core systems. The plant is a digital computer performing high-criticality (or *reactive*) tasks. One high criticality task, BN computation, performs fault diagnosis. We also assume that on this computer, low- and medium-criticality (or *background*) tasks are running and are competing for CPU cycles, memory, and other computer resources.

Control Input: The input to the plant, 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 this work, if $v(k) \geq u(k)$, then processes are terminated until $v(k) < u(k)$. The termination process happens nearly instantaneous with respect to the sampling period.

Disturbance Generation: We model, for simplicity, disturbances to a computer (including user disturbances) as a Poisson process with rate λ. This introduces a stochastic delay in the actuation of the plant in the sense that even if the control input $u(k)$ increases at the next time step, $u(k+1) > u(k)$, the probability of the actually number of processes increasing is low. We model this phenomenon by a stochastic delay in the control input to the plant.

B. Minimum Degree Pole Placement (MDPP) Controller

We use the notation and MDPP derivations defined in [12] for the controller:

$$H(q) = \frac{b_0 q + b_1}{q^2 + a_1 q + a_2}$$

Here there are no zeros canceled; they are given by $q = \frac{-b_1}{b_0}$. The MDPP algorithm proceeds as follows:

Step 1:

$$B = b_0 q + b_1 = B^+ B^- = \frac{B}{A}$$

Given that there is no zero cancellation, $B^+ = 1$. B^+ is monic; B^- is in general not. Since $B^+ = 1$, then $B^- = b_0 q + b_1$. Let n_A denote the degree of polynomial A. We have: $n_A = 2$ and $n_B = 1$, giving

$$B_m = B^- B'_m = B'_m(b_0 q + b_1)$$

B'_m is a constant because it is the same order as B. Here $B_m = b_{m_0} q + b_{m_1}$.

The steady state is given by:

$$\frac{B_m(1)}{A_m(1)} = \frac{B'_m(b_0 + b_1)}{1 + a_{m_1} + a_{m_2}} = 1$$

$$B'_m = \frac{1 + a_{m_1} + a_{m_2}}{b_0 + b_1}$$

Step 2:

$$AR' + B^- S = A_0 A_m$$

$$(q^2 + a_1 q + a_2)R' + (b_0 q + b_1)S = A_0(q^2 + A_{m_1} q + A_{m_2})$$

$$R' = \frac{R}{B^+} = R$$

$$n_{R'} = n_R = n_S = n_A - 1 = 1$$

Verifying compatibility conditions $n_{R'} \leq \max\{n_{a_m} + n_{A_0} - n_A, n_{B^-} - 1\}$ and $n_S < n_A$, we find that $n_R = n_{R'} + n_{B^+}$ and $n_{A_0} = n_A - n_{B^+} - 1$. Deriving our controller u, we have:

$$R' = q + r_1 = R$$

$$S = s_0 q + s_1$$

$$A_0 = q + a_0$$

$$(q^2 + a_1 q + a_2)(q + r_1) + (b_0 q + b_1)(s_0 q + s_1) = A_0(q^2 + a_{m_1} q + a_{m_2})$$

$$R = q + r$$

Last step is to compute T:

$$T = A_0 B'_m = (q + a_0) B'_m$$

And we now have: $u = \frac{T}{R} u_c - \frac{S}{R} y$, our control law.

IV. Experiments

In our experiments, we use data from ADAPT,[2] an electrical power system at NASA Ames, for performing system health diagnosis. ADAPT has abundant data freely available and is representative of many electrical power in aerospace system [15]. We focus on a small part of ADAPT containing five components: a relay, DC load, voltage sensor, current sensor, and relay feedback sensor. For BN inference and computation, we use the open source software package Bayes Net Toolbox (BNT).[3] Experiments were performed on a quad core Intel i5 with 8 GB of memory. All processes created and terminated along with BN inference computation were affixed to a single core (i.e. fixed CPU affinity) to maximize the competition for the CPU. A 5 second pause gap was used between each time step.

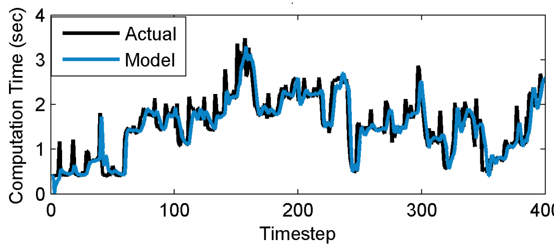

Fig. 2. MRLS estimate of likelihood weighting BN computation time with $\lambda = .94$, $m = .1$.

A. System Identification

To perform system identification, a squarewave representing max number of low-criticality processes $u(t)$ was generated. The squarewave interval size was 20 timesteps and integer values between 0 and 15 inclusive were selected uniformly at random for each interval for $u(t)$. $y(t)$ represents the computation time required for BN inference. Both likelihood weighting and junction tree propagation inference algorithms were used. During the simulation, low-criticality processes were generated via a Poisson distribution with a mean of 15 seconds. Figure 1 shows the squarewave controller simulation. The data $y(t)$, representing the computation time at timestep t was then fitted with first order least squares (LS) and recursive least squares (RLS) using $y(t) = a_1 y(t-1) + b_1 u(t-1)$. The 2nd order RLS model $y(t) = a_1 y(t-1) + a_2 y(t-2) + b_1 u(t-1) + b_2 u(t-2)$ appeared to be typically more resilient to noise.

Figure 2 models the squarewave controller simulation with a forgetting factor λ and MRLS parameter m. The parameters used in Figure 2 provide a good fit and are used in the adaptive MDPP simulation (Section IV-C). The 2nd order LS parameters that best minimized error for estimated likelihood weighting inference computation time were $a_1 = .416, a_2 = .335, b_1 = .010, b_2 = -.064$. This led to the initial conditions of the MDPP controller: $A_{m_1} = -(a_1 + a_2) = -.812$, $A_{m_2} = a_1 a_2 = .194$, and $B_{m_0} = 1 + A_{m_1} + A_{m_2} = .382$. For the MRLS model, we additionally use $\lambda = .93$ and $m = .1$.

B. Naive Controller

To easily visualize how processes are being controlled, a naive 0th order control was implemented (Figure 3). Red circles indicated when a low-criticality process was created pseudo-randomly and green stars indicate that a low-criticality process was terminated. The 0th order controller is unreliable because it terminates processes too easily; there are transient

[2]http://ti.arc.nasa.gov/tech/dash/diagnostics-and-prognostics/adapt-diagnostics/

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

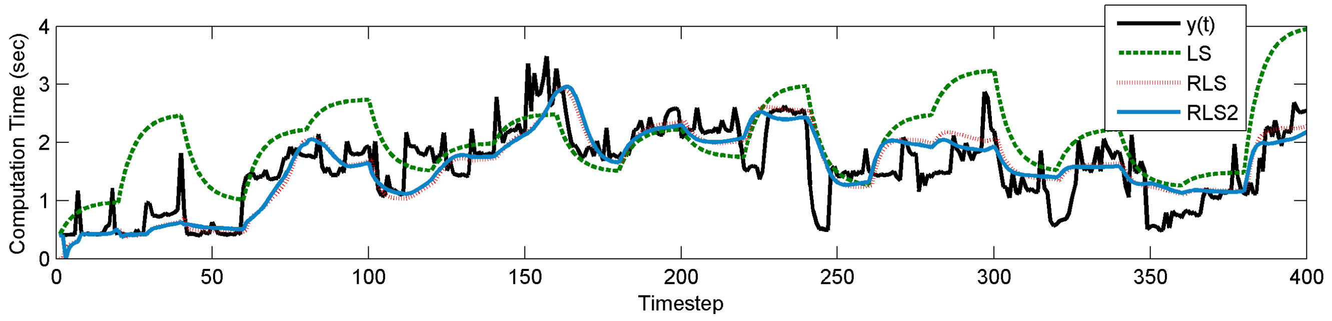

Fig. 1. Likelihood weighting computation time $y(t)$, least squares fit (LS), recursive least squares fit (RLS), and 2nd order RLS fit (RLS2) for a squarewave controller $u(t)$.

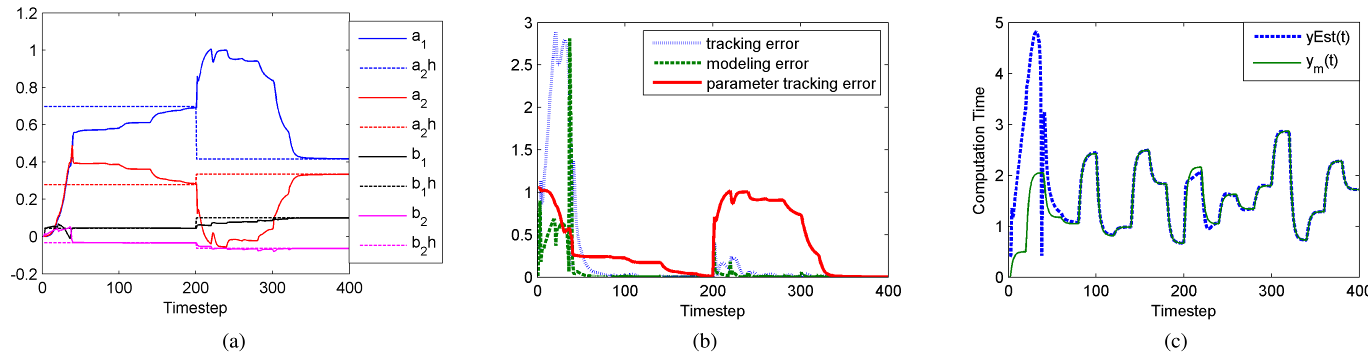

Fig. 4. Adaptive MDPP Control where model is reality (i.e. $\hat{y}(t) = y_m(t)$). At timestep 200, an inference algorithm change from likelihood weighting to junction tree propagation is simulated. (a) shows the parameter values (note: a_1h corresponds to the true parameter $\hat{a}_1$, as well as with $b_1h, \ldots$). (b) shows tracking, modeling, and parameter error. The errors converge to zero before and after the inference algorithm change. (c) shows the adaptive MDPP tracking a squarewave $u_c(t)$. $yEst(t)$ corresponds to $\hat{y}(t)$.

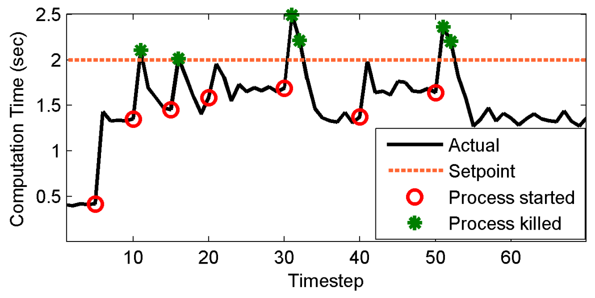

Fig. 3. A naive 0th order controller. A random low-criticality process is terminated when computation time $y(t)$ exceeds a fixed setpoint (2 seconds).

computation time increases when a low-criticality process is generated, but their effect decreases after their first timestep (likely due to the operating system's process creation overhead impeding on the BN inference for a short while). The naive controller is susceptible to noise and terminates an excessive amount of low-criticality processes. The controllers we discuss next are more robust with respect to these issues.

C. Adaptive MDPP Controller – Model Tracking

Using the MRLS and MDPP parameters found in Section IV-A, an adaptive MDPP controller was implemented. First, we look at the case where the model parameters found previously are true and treat them as unknown. MRLS is used to estimate the unknown parameters $\hat{a}_{m_1}, \hat{a}_{m_2}$, and $\hat{b}_{m_0}$ defined by the model $y_m(t) = -\hat{a}_{m_1} y_m(t-2) - \hat{a}_{m_2} y_m(t-1) + \hat{b}_{m_0} u_c(t-1)$ and the MDPP controller selects a $u(t)$ that minimizes error between $y_m(t)$ and $\hat{y}(t)$ for a given $u_c(t)$, where $\hat{y}(t)$ is the estimated computation time. The simulation was run for 400 time steps with an inference algorithm change at timestep 200 (Figure 4). The setpoint $u_c(t)$ was given as the sinusoid $u_c(t) = 3|\sin(t/\pi)| + .5$, which is also used in Section IV-D. The MRLS model converges on the true parameter values perfectly in this simulation; the parameter error converges to zero before and after the inference algorithm change (Figure 4a). Additionally, the modeling and tracking error converge to zero (Figures 4b, 4c). Having confirmed the model performs well in a noiseless environment, we move on to a more realistic simulation.

D. Adaptive MDPP Controller – Real Environment

Applying the parameters found in Section IV-A, we produced an effective controller able to track an inference algorithm and setpoint change (Figure 5). The initial θ parameters were set to the parameters obtained for likelihood weighting via LS in Section IV-A. Despite a simultaneous inference algorithm (likelihood weighting to junction tree propagation) and setpoint change at timestep 200, we see rapid adaptation. The likelihood weighting algorithm requires significantly more computation time than junction tree propagation for this particular BN [16].

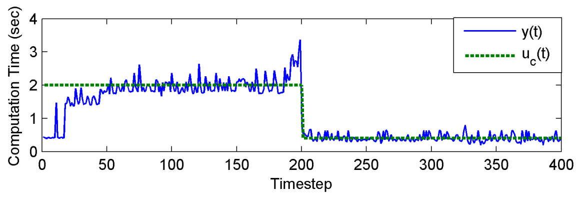

Fig. 5. Adaptive MDPP controller in real environment. The BN inference algorithm is changed from likelihood weighting to junction tree propagation at timestep 200 with an effective adaptation from the MRLS and MDPP. The setpoint $u_c(t)$ was also dropped from 2 to .5 seconds.

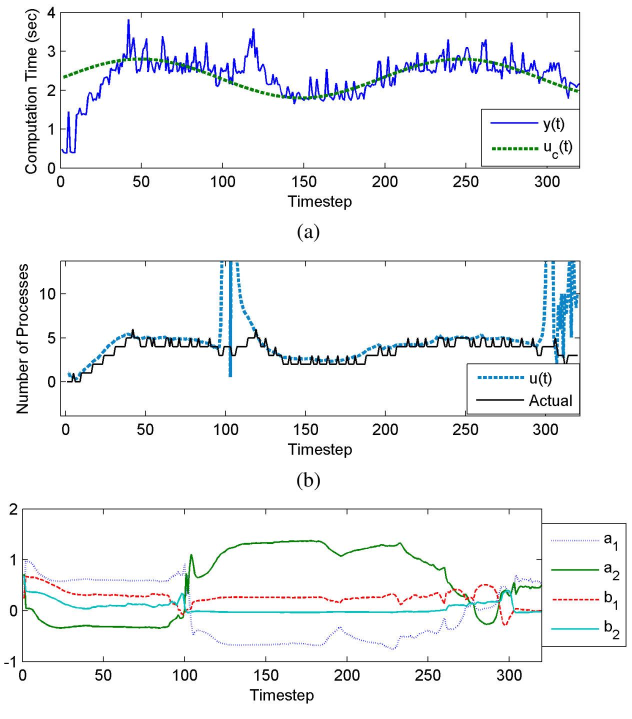

Fig. 6. Adaptive MDPP tracking a sinusoidal $u_c(t)$. (a) shows the computation time $y(t)$ aligned with the setpoint $u_c(t)$. (b) shows how the low-criticality process limit $u(t)$ as well as the actual number of low-criticality processes, which is unknown to the controller and model. The adaptation of the parameter values a_1, a_2, b_1, and b_2 is demonstrated in (c).

Figure 6 incorporates a sinusoidal number of max low-criticality processes, $u(t)$, without an inference algorithm change. 6a shows good tracking and a relatively low tracking error due to noise. 6b shows the actual number of low-criticality processes and how they are affected by $u(t)$. 6c shows parameter adaptation and large amounts of movement, especially near timesteps 100 and 300, which may be due to the low $\lambda = .93$.

V. Conclusion and Future Work

In this work, we have implemented a novel method of controlling Bayesian network inference computation time. Our motivation is to provide soft-realtime (reactive) computation without the use of a specialized hard real-time OS. The use of a Modified Recursive Least Squares with a Minimum Degree Pole Placement controller was effective in tracking inference algorithm changes and responding well to a varying setpoint $u_c(t)$. The controller was able to withstand the noise of the operating system and the unpredictable nature of low-criticality processes. We have only tested on a single device, however; ideally, such inference algorithm control, and broadly AI computation control, should be implemented on a host of devices. Future work will test devices with more exotic hardware, such as mobile devices and high performance clusters, as well as alternative methods to terminating processes–such as suspending and adjusting process priority.

Acknowledgment

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

References

[1] U. Lerner, R. Parr, D. Koller, and G. Biswas, “Bayesian fault detection and diagnosis in dynamic systems,” in *Proceedings of the National Conference on Artificial Intelligence*. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2000, pp. 531–537.

[2] O. J. Mengshoel, M. Chavira, K. Cascio, S. Poll, A. Darwiche, and S. Uckun, “Probabilistic model-based diagnosis: An electrical power system case study,” vol. 40, no. 5, pp. 874–885, 2010.

[3] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury, *Feedback control of computing systems*. Wiley Online Library, 2004.

[4] 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,” *Software Engineering for Self-Adaptive Systems*, pp. 48–70, 2009.

[5] M. Pitarelli, “Anytime decision making with imprecise probabilities,” in *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-94)*, Seattle, WA, 1994, pp. 470–477.

[6] 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*, vol. 50, no. 2, pp. 157–224, 1988.

[7] T. Abdelzaher and N. Bhatti, “Web server QoS management by adaptive content delivery,” in *Quality of Service, 1999. IWQoS’99. 1999 Seventh International Workshop on*. IEEE, 1999, pp. 216–225.

[8] Y. Diao, J. Hellerstein, and S. Parekh, “Optimizing quality of service using fuzzy control,” *Management Technologies for E-Commerce and E-Business Applications*, pp. 42–53, 2002.

[9] 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*, vol. 23, no. 1, pp. 127–141, 2002.

[10] C. Xu, B. Liu, and J. Wei, “Model predictive feedback control for QoS assurance in webservers,” *Computer*, vol. 41, no. 3, pp. 66–72, 2008.

[11] C. Hollot, V. Misra, D. Towsley, and W. Gong, “On designing improved controllers for aqm routers supporting tcp flows,” in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3. IEEE, 2001, pp. 1726–1734.

[12] K. Astrom and B. Wittenmark, *Adaptive control*. Addison-Wesley Longman Publishing Co., Inc., 1994.

[13] J. Monaco, D. Ward, R. Barron, and R. Bird, “Implementation and flight test assessment of an adaptive, reconfigurable flight control system,” in *Proceedings of the 1997 AIAA Guidance Navigation and Control Conference, AIAA Paper*, vol. 97, 1997, p. 3738.

[14] M. Bodson, “An adaptive algorithm with information-dependent data forgetting,” in *American Control Conference, 1995. Proceedings of the*, vol. 5. IEEE, 1995, pp. 3485–3489.

[15] S. Poll, A. Patterson-Hine, J. Camisa, D. Garcia, D. Hall, C. Lee, O. Mengshoel, C. Neukom, D. Nishikawa, J. Ossenfort *et al.*, “Advanced diagnostics and prognostics testbed,” in *Proceedings of the 18th International Workshop on Principles of Diagnosis (DX-07)*, 2007, pp. 178–185.

[16] O. Mengshoel, I. A., and E. Reed, “Reactive Bayesian network computation using feedback control: An empirical study,” in *Bayesian Modeling Applications Workshop at UAI-12*, 2012.