Stochastic System Monitoring and Control*

Gregory Provan†
Rockwell Science Center
1049 Camino Dos Rios, Thousand Oaks, CA 91360
gmprovan@rsc.rockwell.com

Abstract

In this article we propose a new technique for efficiently solving a specialized instance of a finite state sequential decision process. This specialized task requires keeping a system within a set of nominal states, introducing control actions only when forbidden states are entered. Instead of assuming that the process evolves only due to control actions, we assume that system evolution occurs due to both internal system dynamics and control actions, referred to as endogenous and exogenous evolution respectively. Since controls are needed only for exogenous evolution, we separate inference for the case of endogenous and exogenous evolution, obtaining an inference method that is computationally simpler than using a standard POMDP framework for solving this task. We summarize the problem framework and the algorithm for performing sequential decision-making.

1 Introduction

Sequential stochastic decision processes over discrete time are encountered frequently in practice, and one well-known technique to solve them is Partially Observable Markov Decision Processes (POMDPs) [1, 10]. However, inference on POMDPs is computationally very expensive, e.g. computing an optimal policy for a finite horizon is PSPACE-hard [11]; for real-time control tasks this inefficiency can be problematic. In addition, the POMDP framework has a number of strong assumptions, such as system evolution being (implicitly) tied to control actions.[1]

In this article we describe a specialized instance of the POMDP framework that is geared towards efficiently solving stochastic monitoring and control tasks, and propose a technique for solving this task. This specialized task requires keeping a system within a set of steady-state nominal states, introducing control actions only when forbidden states are entered; the *monitoring* phase runs continuously, with the *control* phase invoked only when necessary. To take advantage of this task structure, we relax the assumption that system evolution is (implicitly) tied to control actions (called the implicit-event model in [3]), and the corresponding assumption that a control action is taken at every time step. We instead assume that system evolution occurs due to both internal system dynamics and control actions, referred to as endogenous and exogenous evolution respectively; this model is called the explicit-event model in [3].

We propose a technique for solving this task, in which we separate the representations and algorithms for the cases of endogenous and exogenous evolution. We perform inference on the implicit-event and explicit-event models as appropriate, rather than always solving the implicit-event model (as in the standard POMDP framework), or interleaving the models. We call this approach the *on-demand control approach*. Our proposed inference method is computationally simpler than using a standard POMDP framework for solving this task, with the efficiency gains being proportional to the relative proportions of endogenous and exogenous evolution. In the limit, if we assume that there is no strictly endogenous evolution, then our approach is identical to the standard POMDP approach.

Our contributions are as follows. We provide a formal framework for solving stochastic monitoring and control tasks, in contrast to the *ad hoc* solutions often adopted in approaches that merge condition monitoring with control. For this particular task, we pro-

*Research supported in part by The Office of Naval Research under contract number N00014-98-3-0012.

†Tel: (805) 373-4726.

[1] It is possible to take the null action and allow system evolution due to the internal system dynamics, but here we make it *explicit* that the system's dynamics can alter system state independent of any control actions.

vide a solution method that takes advantage of task characteristics to improve the efficiency of the general POMDP approach. We illustrate our method using a process control example.

The remainder of the article is organized as follows. Section 2 introduces our notation and defines the tasks we are solving. Section 3 describes the inference methods we use. Section 4 outlines the application of this framework to a process-control system. Section 5 compares and contrasts this approach with related material, and Section 6 summarizes our contributions.

2 Task Specification

This section formalizes the general task that we are solving, and then applies this framework to the monitoring and control context.

2.1 Generic Task Specification

We are interested in solving a set of control optimization tasks in which the set of system states S, observations Z and control actions A are specified using a discrete set of possible values. Our general goal is to drive the system though a sequence of such states while minimizing some criterion, such as energy expenditure. The goal that is the focus of this paper is that of *steady state monitoring and control*: we wish to maintain a system in a steady-state nominal condition without entering into particular forbidden states; in case a forbidden state is entered, a sequence of control actions is executed to drive the system back into a nominal steady-state. Within this framework, there are two main tasks: (1) monitoring that the system is in a steady-state nominal condition; and (2) executing controls to drive the system back to a steady-state nominal condition if a forbidden state is entered. Note that task (1) involves *no* control actions, but rather state classification.

We start by presenting the standard notation for such control optimization tasks, where we assume that state transitions occur only due to control actions.[2] For each state $s_i \in S$ for which the agent selects a control $a \in A$, the agent receives an immediate reward with expected value $\varphi(s_i, a)$, at which point the system makes a transition to state $s_j \in S$ with probability $Pr(s_j|s_i, a) \in [0, 1]$, and the agent makes an observation $z_j \in Z$ with probability $Pr(z_j|s_j, a) \in [0, 1]$.

The actual system state cannot be directly observed, but its probability is inferred from the observations Z and control actions A. Let Σ denote a vector of state probabilities, called a belief state, where $\Sigma(s_i)$ denotes the probability that the system is in state s_i. If action a is taken and observation z follows, the successor belief state, denoted Σ^a_z, is determined by revising each state probability as follows:

$$\Sigma^a_z(s_j) = \frac{Pr(z|s_j, a) \sum_{s_i \in S} Pr(s_j|s_i, a)\Sigma(s_i)}{Pr(z|\Sigma, a)}, \quad (1)$$

where the denominator is a normalizing factor:

$$Pr(z|\Sigma, a) = \sum_{s_j \in S} Pr(z|s_j, a) \sum_{s_i \in S} Pr(s_j|s_i, a)\Sigma(s_i).$$

Our task is to select a sequence of actions, called a policy, that optimizes a performance criterion V. We summarize this implicit-event model using $\mathcal{M}_I = \langle S, A, Z, V, \Sigma \rangle$. In many cases, the objective is to maximize the expected total discounted reward over an infinite horizon, given a discount factor β. Typically, value iteration is performed over the one-step function for each belief state Σ:

$$V'(\Sigma) = \max_{a \in A} \left[\sum_{s \in S} \Sigma(s_i)\varphi(s_i, a) + \beta \sum_{z \in Z} Pr(z|\Sigma, a) V(\Sigma^a_z) \right]. \quad (2)$$

It is well-known that value iteration coverges to the optimal value function in the limit.

2.2 On-Demand Control Specification

The generic (implicit-event) approach makes two strong, related assumptions. The first strong assumption is that the system makes transitions only due to control actions, i.e. actions exogenous to the system. An alternative view of this assumption is that effects of system-internal (or non-control) events are folded into the transition probabilities associated with the action [3]. The second assumption is that there is a control action at every time step. Together, these assumptions require a computational model in which effects of actions must be evaluated to compute the value function.

In our new formulation, we relax both of these assumptions. In contrast to assuming only exogenous transitions, we distinguish exogenous transitions from system-internal, or endogenous, transitions. In other words, the system evolves over a set of discrete time steps due to both endogenous and exogenous events; we assume that only one action can take place at any given time. To specify this notion, we define exogenous actions (events) $a \in A$ and endogenous events $\gamma \in \Gamma$. This results in state transitions from $s_i \in S$ to $s_j \in S$ that occur either with probability $Pr(s_j|s_i, a) \in [0, 1]$ (exogenous transition) or $Pr(s_j|s_i, \gamma) \in [0, 1]$ (endogenous transition), at which point the agent makes an observation $z_j \in Z$ with probability $Pr(z_j|s_i, a) \in [0, 1]$

[2] We adopt the notation introduced in [8].

(resp. $Pr(z_j|s_i,\gamma) \in [0,1]$). We summarize this implicit-event model using $\mathcal{M}_I = \langle S, \Gamma, Z, V, \Sigma \rangle$.

We relax the second assumption, since in many real-world applications (loosely defined under monitoring and control), an agent does not apply controls to the system continuously, but only in particular states. For example, a control loop may monitor a process control system, taking control actions only when the fluid pressure deviates from the nominal value. Note that we are really interested in distinguishing the equivalence classes of nominal states and forbidden states.

To formalize this notion, we partition the system state S into two disjoint sets, S_y and S_n, such that $S = S_y \cup S_n$ and $S_y \cap S_n = \emptyset$; S_y is the subset of forbidden states for which control actions are necessary, and S_n is the subset of states for which no control actions are necessary. In other words, as long as the system is in some state $s \in S_n$, no control actions are taken: this is the *monitoring phase*. Once the system enters into some forbidden state $s \in S_y$, a set of control actions are taken: this is the *control phase*.[3]

There are now two forms of state evolution, those due to endogenous and those due to exogenous events, respectively denoted by the following reformulations of Equation 1:

$$\begin{aligned}\Sigma_z^{\gamma}(s_j) &= \frac{Pr(z|s_j,\gamma)\sum_{s_i \in S} Pr(s_j|s_i,\gamma)\Sigma(s_i)}{Pr(z|\Sigma,\gamma)},\\ \Sigma_z^{a}(s_j) &= \frac{Pr(z|s_j,a)\sum_{s_i \in S} Pr(s_j|s_i,a)\Sigma(s_i)}{Pr(z|\Sigma,a)}.\end{aligned}$$

Note that we could also represent the endogenous evolution case without using control actions, but purely in terms of probabilistic transitions, such as using Markov transition matrices or a Bayesian network.

2.3 Model Specification

We adopt the standard control-theoretic decomposition of a system into plant and control models:

1. Plant Model: this specifies the physical behavior of the system, i.e., the control to sensor (observable) mapping: $f : A \mapsto Z$;
2. Control Model: this specifies the control behavior of the system given the sensor values (with no feedback from the plant model), i.e., the sensor to control mapping: $g : Z \mapsto A$.

By composing these two models, we can specify the feedback control behavior of the system based on the system behavior defined by the plant network. The plant model corresponds to our endogenous model, and the composite model to our exogenous model.

Distinguishing endogenous and exogenous evolution can result in a larger model that is more complicated to evaluate. For example, one could interleave the endogenous and exogenous models, and place a number of contraints over this interleaved model [3].

Rather than adopt an approach like this, we tailor to the task structure a simpler model: for the monitoring phase we use a model that is a simple extension of the plant model, and for the control phase we augment this monitoring model with additional control structure. These two models may differ in the events and value functions defined. For example, different value functions are used in the two phases. The value function for the monitoring phase is specified *based on state values only*, and is independent of the exogenous control action. This means that the expected value can be computed relatively efficiently, since optimization over the set of all possible control actions is unnecessary. To see this greater simplicity, compare the monitoring value function of Equation 3 to the control value function of Equation 2.

$$V'(\Sigma) = \left[\sum_{s \in S} \Sigma(s_i)\varphi(s_i,\gamma)\right]. \qquad (3)$$

From Equation 2 the expected value for the control phase (using the standard POMDP computation) is computed based on both the system state and the control actions.

Our main interest during the monitoring phase is estimating the probability distributions over the operating modes of particular system components. We use $\mathcal{D}$ to denote the set of system components of interest. An unobservable variable, called an assumable, is associated with every component whose operating mode (or health state) we want to diagnose, and it specifies the operating modes of its associated component, such as nominal, overdrive and broken.

3 Modeling and Inference Method

This section maps our task framework into a computational framework.

3.1 Model Specification

We specify our system as an Influence Diagram (ID) [12]. An influence diagram consists of a factored probability distribution, which is represented as a Bayesian network [12], together with a set of decision nodes and value nodes. A Bayesian network (BN) consists of a directed acyclic graph $G(X, E)$ of nodes X and edges

[3]We assume that there could be either a sequence of control actions, or a single action, followed by a return to the monitoring phase.

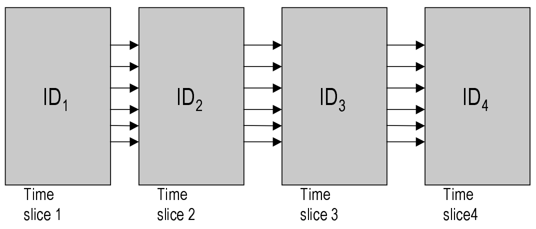

Figure 1: Structure of Generic ID with 4 time slices. Each block represents a time-slice model, with directed edges joining adjacent time-slices

$E \subseteq X \times X$. The graph specifies the probability factoring as follows: the joint probability is given by the product of the probabilities of each $X_i \in X$ conditioned on its parent variables in the graph $pa(X_i)$, i.e.

$$Pr(X) = \prod_i Pr(X_i|pa(X_i)).$$

There is a large body of work on computing maximal expected values for IDs–many are reviewed in [3]. We adopt a standard algorithm, as defined in [14].

We represent the sequential decision aspect of our task by explicitly denoting the discrete time indices of each variable in our model. We call this temporal model a Temporal ID, or TID. Through this process, we can identify a sequence of time slices, where we represent the time slice for time t with an influence diagram where all variables are indexed by t and all arcs in this ID are synchronous, i.e., the arcs join nodes with the same temporal index. A sequence of time-slice IDs has a set a diachronic arcs joining nodes with time index t to nodes with time index $t + 1$. Figure 1 shows the structure of a generic ID with 4 time slices.

In our ID model, we use a value node for the optimization criterion, decision nodes for events, and chance nodes for all other system entities. If we use the standard implicit event model, then we obtain an ID as shown in Figure 2. We represent an ID using a DAG with oval nodes to depict chance variables, square nodes to depict action variables, and diamond nodes to depict value functions. In Figure 2 we denote a single assumable as the node labeled Dx, three sensor nodes Z_1, Z_2, Z_3, three event (decision) nodes A_1, A_2, A_3, and a single value node. Note that the three event nodes directly influence the topmost three nodes in the suceeding time slice.

Figure 3 compares the explicit-event model we use for monitoring with the implicit-event model we use for control Note that the monitoring model now has a single decision node that has as predecessor nodes the assumable node Dx and the three sensor nodes; using this structure, this decision node determines if a forbidden state has been reached (in expectation) based on its predecessors.

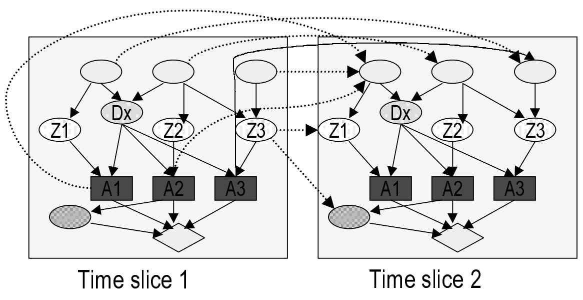

Figure 2: Structure of Generic Implicit-Event model with 2 time slices. We denote the diachronic arcs using dotted lines, and the synchronous edges using solid lines.

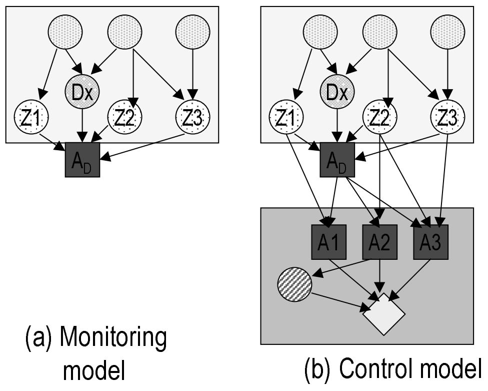

Figure 3: Comparison of Implicit-Event and Explicit-Event models.

During monitoring, for every time-slice we input evidence $\mathbf{e}$ consisting of the control-node and sensor-node values. For some threshold probability p^*, set ς of forbidden values of node Dx, and evidence $\mathbf{e}$ in the network, the decision node A_D takes on values given by

$$A_D = \begin{cases} monitor & \text{if } Pr(Dx = \varsigma|\mathbf{e}) \leq p^* \\ control & \text{otherwise.} \end{cases} \tag{4}$$

This decision node can then be used to invoke the control model to drive the system back into a non-forbidden state when $[A_D = control]$. Note that, using an approach described in [15], we could alternatively model this decision node as a chance node.

So if we are performing only monitoring, we would create a model that has a TID consisting of a sequence of monitoring time slices, as shown in Figure 4. Using our on-demand control representation, we build up a model consisting of time-slice models as required,

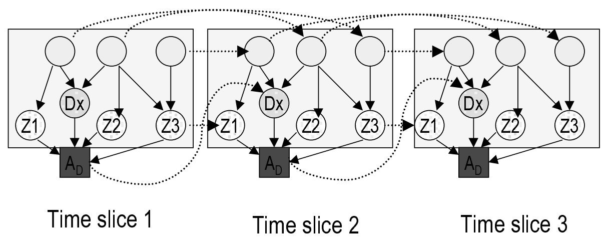

Figure 4: Pure monitoring using Monitoring Explicit-Event models.

i.e., we introduce control time-slices only when necessary. For example Figure 5 shows a case where the first monitoring time slice indicates that a forbidden state has been entered, requiring a sequence of control time-slices, which determine the optimal sequence of actions required to drive the system out of the forbidden state set. Once this has been accomplished, the only model that is needed is a monitoring model, until another forbidden state is entered.

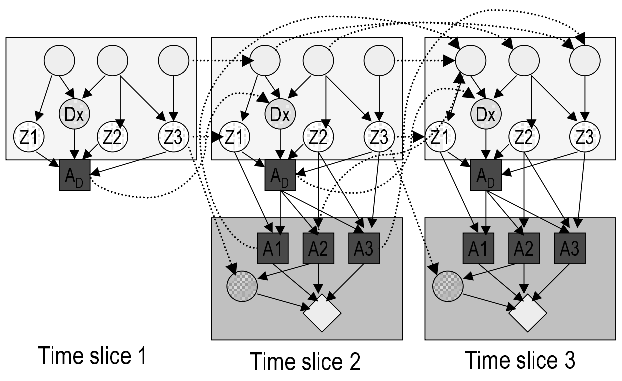

Figure 5: Structure of on-demand Implicit-Explicit model with 3 time slices. The first time-slice involves monitoring, followed by two time-slices for control

3.2 Inference Algorithm

We solve this task using a two-step procedure, using the steps of *estimation* and *control optimization.*

State Estimation this task computes the probability distribution of the assumables given the controls and sensors as evidence. We need to use only the plant Bayesian network for this task, treating the control nodes not as decision nodes but as observable evidence nodes with fixed values. We compute the monitoring value function (Equation 3): if this value function indicates that the probability of an undesirable state $s \in S_y$ exceeds a threshold, the system switches to Control Reconfiguration mode.

Control Optimization this task initializes the assumables as evidence and computes the optimal (stochastic) control. We must use a feedback control network for this task, which includes the plant network integrated with the control sub-network. In this phase a set of optimal controls is derived to drive the system back to a desirable state $s \in S_n$, and the cycle repeats.

We adopt a two-step procedure because these two computations are typically quite different. In the first case, we have no control over the endogenous events, but we instead merely want to monitor these actions. In the second case, we are actually sending controls to the system and want to optimize the value of this sequence of controls.

For computational efficiency, several approaches can be used to take advantage of the problem structure. We outline two here. First, it is possible to have different models and inference algorithms for the two phases. In our implementation, we model system dynamics in the monitoring phase using a Bayesian network, which stochastically estimates assumable states given a sequence of sensor data. Given an output of $\hat{\Sigma}$, we perform the test $\hat{\Sigma}(S_y)$; As long as the test is false, i.e., the assumable state is not forbidden, the estimation cycle continues. Whenever the forbidden state is entered, then the control phase begins. We extend the Bayesian network model to a TID for the control reconfiguration phase. The TID is used to generate the least-cost set of control actions that will drive the system back to a non-forbidden assumable state. After executing this stage for a number of time-slices determined by the same test $\hat{\Sigma}(S_y)$, the system returns to the monitoring phase again. Hence in the monitoring phase we are merely updating the probabilities in the Bayesian network, and in the control reconfiguration phase we are computing on optimal decision sequence in the Influence diagram. Figure 6 depicts how we could organize inference using two separate modules.

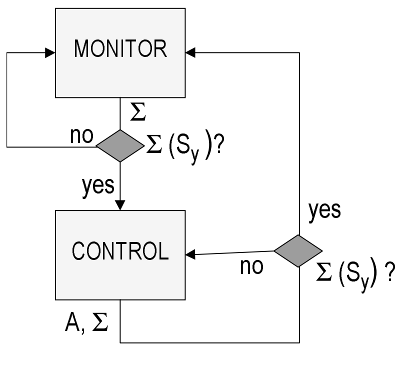

Figure 6: Inference using two disjoint modules, one for monitoring and one for control.

Second, it is possible to have a single model, but to use a control method to focus inference on the monitoring part of the model, and use the full POMDP model only when necessary. Figure 5 depicts the kind of network that would be generated on demand, given a sequence of observable data. This approach can be implemented by taking advantage of structural properties of observations in POMDPs [6].

3.3 Complexity Considerations

This section briefly discusses the efficiency gains obtained by modeling and solving systems using this new approach versus modeling and solving systems using POMDPs. Inference on POMDPs is computationally very expensive, e.g. computing an optimal policy for a finite horizon of $\mathcal{T}$ is worst-case exponential in both $\mathcal{T}$ and the size $\mathcal{S}$ of the action transition matrix [11]. The complexity of the on-demand approach is dependent on the relative frequency of monitoring and control phases. On the one extreme, with all control we obtain a complexity identical to that of the traditional POMDP approach; the greater the degree of monitoring the greater the corresponding increase in efficiency over the traditional POMDP approach. This is because the complexity of monitoring is independent of horizon $\mathcal{T}$, but is worst-case exponential in particular graph parameters, such as the size of the largest clique in a clique-tree representation of the graph, or in the size of the graph-width [7]. The monitoring phase does not perform value maximization, but instead computes the posterior probability of the assumables. Given that the monitoring network is smaller than the control network and shares much of the same structure, it is likely that evaluating a single time-slice of the monitoring network is computationally simpler than that of the single time-slice control network. In real-world applications where device failure is unlikely, the bulk of time is spent in the monitoring phase, rendering this on-demand approach significantly more efficient that the standard POMDP approach for this class of applications.

4 Example

This section applies our approach to a monitoring and control example, that of a motor/pump loop. This loop is a simplification of a large class of process-control loops common to many types of industry, such as chemical processing.

4.1 Pump Loop Description

Figure 7 shows an example of a pump loop, which consists of a fluid tank, a pump connected to a motor, and a pipe loop through which the liquid is pumped. In this pump loop, water flows from the tank to the inlet of the pump and returns to the tank from the outlet of the pump. Of particular interest to this example is the pump inlet and outlet valves used to control the water flow. The water pressure at both the inlet and the outlet of the pump are measured using two pressure sensors. A turbine flowmeter is used to measure the water flow.

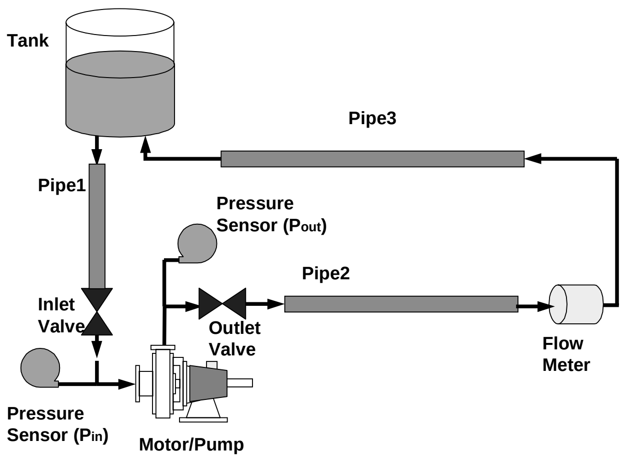

Figure 7: Pump Loop example.

We select a simplified version of this pump-loop, in order to focus on the techniques rather than the details of the example. In particular, we assume we want to control the pressure through the system by controlling the pressure throughput on the pump. We assume that the motor driving the pump has two speeds, normal and high, and that we can set the inlet valve for the pump to four degrees of opening, namely {closed, 50%, 75%, open}.

We are mainly interested in a particular fault that commonly occurs in pump loops: pump cavitation. We assume that the pump can be in one of three operating modes, normal, overdrive and cavitating. Normal and overdrive constitute nominal operating modes. The cavitating mode is one in which pressure imbalances within the pump create bubbles, which are transported with the fluid flow from the low pressure region to the high pressure region in the pump chamber, where they collapse violently and cause problems [4]. Cavitation can be a serious problem resulting in early failure of pumping systems today. This undesirable phenomenon could lead to increased volumetric losses, high noise levels, early failure of pump seals, and severe material damages to pumps.

Given this possibility of cavitation, our control task is to maintain the pressure within the pump loop under nominal system modes, and then reduce the pump speed (and fluid flow) if cavitation occurs, in order to restore system operation to non-cavitating conditions.

We create a system model as follows. We describe two components in our model: a motor/pump and an inlet valve, together with the following sensors: inlet and outlet pressure and pump vibration. The combination of pressure and vibration sensors allows us to distinguish cavitating from nominal conditions [13]. We assume that we can measure the values of pressures P_{in} and P_{out}, and can control the values of input-valve control actuator Ψ_V and the motor-pump actuator, Ψ_{MP}. We assume that we have (unmeasureable) inlet and outlet pressures, P_{in} and P_{out} repectively, and a mode variable for the motor-pump, MP, which takes on values {normal, overdrive, cavitating}. We also have inlet and outlet pressure sensors, $Z_{P_{in}}$ and $Z_{P_{out}}$ repectively.

The controller for a simple valve-pump system operates as follows: given an input flow with pressure P_{in}, this controller governs the actuator settings of valve Ψ_V and pump Ψ_{MP} to maintain an output pressure P_{out} such that $P_{min} \leq P_{out} \leq P_{max}$.

Figure 8 depicts the structure of the plant model. This network depicts the physical inter-relationships for the pump-loop, without specifying any actions or value functions. To specify this network, we need to identify prior distributions on Ψ_V, Ψ_{MP} and MP, as well as the conditional distributions $Pr(P_{out}|P_{in}, A_{MP}, MP)$, $Pr(Z_{P_{in}}|P_{in})$, and $Pr(Z_{P_{out}}|P_{out})$.

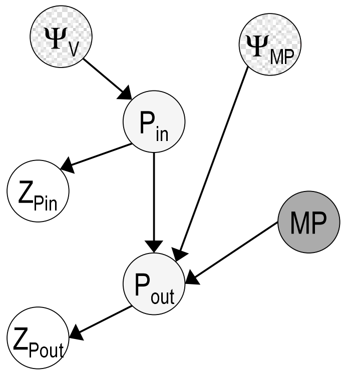

Figure 8: Plant network.

Figure 9 depicts the structure of the control model for two time slices. This figure uses much of the structure of the plant model, except that we now define event nodes for (1) whether we do monitoring or control (A_M), (2) setting the valve (A_V), and (3) setting the motor-pump (A_{MP}). Note that the event A_V (resp. A_{MP}) at time t then governs the setting of the corresponding actuator Ψ_V (resp. Ψ_{MP}) at time $t+1$.

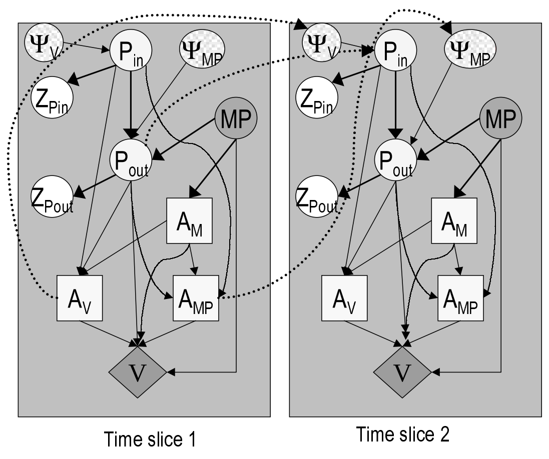

Figure 9: Feedback control network.

In our process control example, during the monitoring phase we set the threshold probability for entering forbidden states, i.e., the p^* of Equation 4, to 0.66. This computation requires only state estimation. In contrast, for the control reconfiguration phase we compute the expected value function based on both the system state and the control action sequence that must be taken to ensure that P_{out} returns to its nominal range. In this case we have a value function $V(s_i, a)$ that is based on state values for input and output pressures, and mode settings of valve and pump, as well as settings for the actions taken given the particular state.

4.2 Results

We ran some experiments with this simple pump-loop example to test the relative time and space requirements for the standard (POMDP) approach and the on-demand control approach. Running the system for 10 and for 20 time steps, we varied the initial actuator settings to generate scenarios with different percentages of steps consisting of monitoring. Note that the POMDP and On-Demand approaches are identical in the case of 0% monitoring.

Table 1 summarizes data from these experiments averaged over 10 runs. Space requirements grow roughly linearly with $\mathcal{T}$, and decrease roughly linearly with increases in percentage of monitoring. For the temporal Bayesian network used for monitoring, the worst-case time complexity is independent of $\mathcal{T}$, and the space complexity is linear in $\mathcal{T}$, as indicated by the data. In terms of time, however, the percentage of monitoring significantly reduces the time, and the time requirements increase roughly exponentially with $\mathcal{T}$ when optimal control is computed.[4]

% Monitoring	Space (Kb)		Time (s.)	
	$\mathcal{T}$=10	$\mathcal{T}$=20	$\mathcal{T}$=10	$\mathcal{T}$=20
0	511	988	191.10	*
20	371	723	37.31	135.54
50	213	422	4.96	23.12
100	136	289	1.19	1.27

Table 1: Comparative results for On-Demand Approach on the simple pump-loop. Data is presented for fixed horizons of $\mathcal{T}$=10 and 20.

5 Relation to Other Work

There is a great deal of work in POMDPs and probabilistic planning that use approximation approaches. Rather than use approximation algorithms, e.g. [2], other special-case algorithms [8], or domain-independent hierarchical abstractions [9], we introduce a structural "approximation", namely tailoring the problem structure to the actual task being performed. This approach is not as general as the above-mentioned approaches, but it produces a relatively efficient solution to the problem class being addressed.

This work also bears some resemblance to to stochastic automata [5]. In the area of formal methods, a number of stochastic automata or stochastic process algebra models have been proposed, e.g. [5]. These approaches are concerned primarily with specification, and only secondarily with computational issues. In contrast, we focus on both aspects.

6 Summary and Conclusions

We propose a technique for creating on-demand models tailored to the task of monitoring and control. This approach is computationally more efficient than the traditional POMDP approach, which solves a control optimization task over the entire time horizon necessary. Our approach breaks up the inference into two steps, estimation and control optimization, and solves the simpler estimation task when control optimization is not necessary. The greater the percentage of time that only state estimation is required, the greater is the computational advantage of using the proposed approach over the POMDP approach; if control optimization is required all the time, the proposed approach is identical to the POMDP approach.

[4]The * indicates that the run did not run to completion after 20 hours.

References

[1] K.J. Astrom. Optimal control of markov decision processes with incomplete state estimation. *Journal of Math. Annal. Appl.*, 102:174–205, 1965.

[2] C. Boutilier, R. Brafman, and C. Geib. Structured reachability analysis for markov decision processes. In *Proc. Intl. Conf. on Uncertainty in Artificial Intelligence (UAI)*, pages 24–32, 1998.

[3] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of AI Research*, 11:1–94, 1999.

[4] C.E. Brennen. *Cavitation and Bubble Dynamics.* Oxford University Press, Oxford, England, 1995.

[5] P.R. D'Argenio, J.-P. Katoen, and E. Brinksma. A compositional approach to generalised semi-markov processes. In *Proceedings of the 4th International Workshop on Discrete Event Systems, WODES'98,* Caligari, Italy, pages 391–387. IEE, 1998.

[6] Adnan Darwiche and Gregory Provan. The Effect of Observations on the Complexity of Model-Based Diagnosis. In *Proc. AAAI National Conference.* Morgan-Kaufmann Publishers, 1997.

[7] R. Dechter. Bucket Elimination: A unifying framework for Reasoning. *Artificial Intelligence Journal*, 107, October 1999.

[8] E. Hansen. Solving POMDPs by Searching in Policy Space. In *Proc. Intl. Conf. on Uncertainty in Artificial Intelligence (UAI)*, 1998.

[9] M. Hauskrecht, N. Meuleau, L. Kaelbling, T. Dean, and C. Boutilier. Hierarchical Solution of Markov Decision Processes using Macro-actions. In *Proc. Intl. Conf. on Uncertainty in Artificial Intelligence (UAI)*, pages 220–229, 1998.

[10] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence Journal*, 101, 1998.

[11] C. Papadimitriou and J. Tsitsiklis. The Complexity of Markov Chain Decision Processes. *Mathematics and Operations Research*, 12(3), 1987.

[12] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, 1988.

[13] G. Provan and Y.-L. Chen. Component-based modeling and diagnosis of process-control systems. In *Proc. 1999 IEEE Intl. Symposium on Computer-Aided Control System Design*, pages 194–199, Kohala Coast, HI, August 1999.

[14] J. Tatman and R. Shachter. Dynamic Programming and Influence Diagrams. *IEEE Trans. Systems, Man and Cybernetics*, 20:365–379, 1990.

[15] N. Zhang. Probabilistic Inference in Influence Diagrams. In *Proc. Intl. Conf. on Uncertainty in Artificial Intelligence (UAI)*, 1998.