Approximate Dynamic Programming Based Optimal Control Applied to an Integrated Plant with a Reactor and a Distillation Column with Recycle

Thidarat Tosukhowong and Jay H. Lee
School of Chemical and Biomolecular Engineering, Georgia Institute of Technology,
311 Ferst Dr., Atlanta, GA 30332

DOI 10.1002/aic.11805
Published online February 20, 2009 in Wiley InterScience (www.interscience.wiley.com).

An approximate dynamic programming (ADP) method has shown good performance in solving optimal control problems in many small-scale process control applications. The offline computational procedure of ADP constructs an approximation of the optimal "cost-to-go" function, which parameterizes the optimal control policy with respect to the state variable. With the approximate "*cost-to-go*" function computed, a multi-stage optimization problem that needs to be solved online at every sample time can be reduced to a single-stage optimization, thereby significantly lessening the real-time computational load. Moreover, stochastic uncertainties can be addressed relatively easily within this framework. Nonetheless, the existing ADP method requires excessive offline computation when applied to a high-dimensional system. A case study of a reactor and a distillation column with recycle was used to illustrate this issue. Then, several ways were proposed to reduce the computational load so that the ADP method can be applied to high-dimensional integrated plants. The results showed that the approach is much more superior to NMPC in both deterministic and stochastic cases.
© 2009 American Institute of Chemical Engineers *AIChE J,* 55: 919–930, 2009

Keywords: approximate dynamic programming, nonlinear optimal control, integrated plant, dynamic optimization, nonlinear model predictive control

Introduction

Currently, the most commonly used approach to solve nonlinear optimal control problems in process industries is the nonlinear model predictive control (NMPC) method. In this approach, a dynamic model and online state estimates are used to build a prediction of the future output of the plant, based on which an online optimizer finds a sequence of manipulated input values that optimize a chosen objective function subject to specified constraints. Since new measurements can improve the state estimates at next sample time, only the inputs for the current sample time are implemented and the optimization is repeated. Despite the popularity of this approach, there are two limitations of the NMPC approach. First, the online computational load for solving a nonlinear optimization problem at each sample time can be significant when applied to a high-dimensional integrated chemical plant. Model for such a plant often takes the form of stiff and high-dimensional ordinary differential equations (ODEs), or differential algebraic equations (DAEs). In addition, the common presence of long transient dynamics in

Current address of T. Tosukhowong: Eastman Chemical Company, P.O. Box 1972, Kingsport, TN 37662-5150.
Correspondence concerning this article should be addressed to J. H. Lee at jay.lee@chbe.gatech.edu.

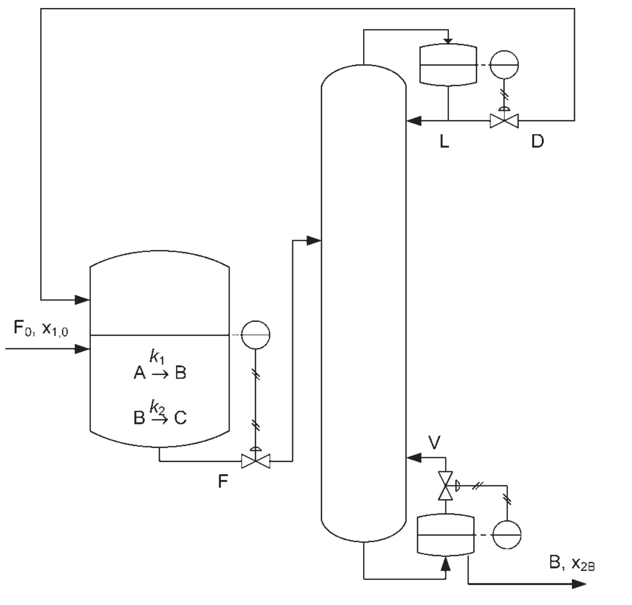


Figure 1. Process schematic of the reactor of the reactor-distillation integrated plant.

such a plant demands the use of long prediction and control horizons to ensure satisfactory performance. These factors combine for a large, difficult optimization problem to be solved online. The second limitation of this framework is in the handling of uncertainty. Uncertainty is a crucial part of a plant-wide real-time optimization problem in process industries. However, the current mathematical programming based approach mainly solves at each sample time a *deterministic open-loop* optimal control problem, based on a *best* prediction of future trajectories, and, hence, it does not consider stochastic variations or ameliorating effect of future feedback measurements. As a result, NMPC is inherently a suboptimal method for feedback control of stochastic systems.

Alternatively, both issues can be addressed by the approximate dynamic programming (ADP) approach. This approach tries to build offline an accurate approximation of so called "cost-to-go" or "value" function for a multistage (typically, infinite horizon) dynamic optimization. The resulting "cost-to-go" function of ADP can be used to reduce the online multistage optimization problem into an equivalent single-stage problem. As a result, the online computational load dramatically decreases. In the process control community, applications of this approach are still in a nascent stage and have been limited to problems with small state and action space. The current state of the art of this approach presented by Lee and coworkers[1,2] has been developed to solve several optimal control problems with small action space. When the process model is highly nonlinear and is of high-order, the offline computational load to derive the cost-to-go function according to this method can be too high. As a result, the objective of this article is to propose ways to reduce the computation to a level feasible for an integrated plant under uncertainties.

The remainder of this article is organized as follows. In the second section, we introduce a case study of a reactor and a distillation column, with recycle as a representative example of a typical integrated plant with high-dimensional state and action space. NMPC is applied to this problem in the third section to establish a base case for performance comparison. The fourth section introduces the ADP framework, and identifies computational barriers associated with its offline cost-to-go learning procedure. Our contribution is to propose systematic approaches to overcome existing problems in ADP learning procedures. Then, we show how to apply ADP to an integrated plant, which results in superior performances compared to NMPC approaches in both deterministic and stochastic cases. Finally, the conclusions are presented in the Conclusions section.

Case Study: A reactor and a distillation column with recycle

In this study, we use a reactor-distillation-recycle system shown in Figure 1, which was first studied by Kumar and Daoutidis,[3] to represent an integrated plant with a material recycle. This system is described by 39 nonlinear ODEs. The concentrations are denoted by x_{ij}, where i indicates chemical species (1: reactant A, 2: desired product B, 3: undesired product C), and j represents processes (R: reactor, D: condenser, B: reboiler, 0: feed, 1–15: tray location). We define four operating modes for this system as shown in Table 1. Nominal values of process variables in the operating mode 1 are shown in Table 2. The feed compositions of component A ($x_{1,0}$) and component B ($x_{2,0}$) are 1.0 and 0.0, respectively. There are six manipulated variables for plant-wide optimization including the feed (F_0), the holdup set points (i.e., M_R^{sp}, M_D^{sp}, M_B^{sp}), the reflux flow rate (L), and the production rate (B). This system is very ill-conditioned because of the large recycle stream that causes a two-time scale behavior. The residence times of the reactor and on each tray of the distillation column are on the order of a few minutes. However, the transient dynamics of the entire plant last 2–3 days after a step change. The objective of the optimal control is to maneuver the system from the nominal operating mode 1 to other modes, while keeping the manipulated and the output variables within their operating limits as shown in Tables 3 and 4, respectively.

Nonlinear Model Predictive Control (NMPC) Approach

Recently, nonlinear dynamic optimization has gained a wide interest in the process system community. In particular, the nonlinear model predictive control (NMPC) method is currently the norm for advanced process control of nonlinear systems. In this method, an optimal control problem over a moving time window is cast as a nonlinear program (NLP), based on a nonlinear dynamic model and the information of the state, and the disturbance up to the current time. Then, an online optimizer is used to find an optimal input profile and only the inputs for the current sample time are imple-

Table 1. Operating Modes of the Reactor-Distillation Integrated Plant

Specifications	Mode 1	Mode 2	Mode 3	Mode 4
Product composition x_{2B}	0.886	0.85	0.905	0.82
Production rate B	100	115	85	125

Table 2. Nominal Values for the Process Variables of the Reaction-Distillation Integrated Plant Example

Feed flow rate (F_0)	100 hr^{-1}	Reactor outlet flow rate (F)	1880 hr^{-1}
Recycle flow rate (D)	1780 hr^{-1}	Reflux flow rate (L)	290 hr^{-1}
Vapor boilup rate (V)	2070 hr^{-1}	Product flow rate (B)	100 hr^{-1}
Kinetic rate 1 (k_1)	1	Kinetic rate 2 (k_2)	1
Volatility constant 1 (α_1)	4	Volatility constant 2 (α_2)	2
Reactor holdup (M_R)	110	Condenser holdup (M_D)	173
Reboiler holdup (M_B)	181	Tray liquid holdup (M_T)	175
x_{1R}	0.8996	x_{2R}	0.0939
x_{1D}	0.9496	x_{2D}	0.0494
x_{1B}	0.0104	x_{2B}	0.8863

mented. There are currently many algorithms used to solve NLP, such as a successive quadratic programming (SQP) method, and simultaneous optimization approaches. The computational complexity of these approaches is a function of the number of states (n_x), outputs (n_y), inputs (n_u), prediction horizon (p), and control horizon (m).

Although development of sophisticated NLP algorithms may allow larger and larger nonlinear optimal control problems to be solved in real-time, the problem solved at each sample time is still a *deterministic* open-loop optimization problem, which does not consider the future uncertainties or feedbacks. In addition, even though the receding horizon implementation of the NMPC approach attempts to address the uncertainties by incorporating the output mismatch into the state estimate and resolving the optimization problem at every sample time, the solution of this approach can still be highly suboptimal. Notable examples can be found in Lee et al.[1,4]

In the following subsection, the base case deterministic control study with a NMPC controller that uses a full-scale nonlinear model and a conventional sequential optimization approach is presented to examine the computational load. Then, we introduce a nonlinear model reduction technique to improve the computational time of NMPC. Although, the model reduction technique may not drastically reduce the online optimization time of NMPC, it is a very important tool to be used with the ADP method in a later section to improve the offline computational load of the ADP. The performance of NMPC under stochastic uncertainties will be presented later in the subsection entitled "*Results of ADP application to a stochastic nonlinear optimal control problem.*"

Performance of NMPC in deterministic scenario

In this section, the NMPC is used to perform the grade transition. We chose the sequential quadratic programming (SQP) method to solve the optimization for the NMPC in MATLAB™ 7.0 environment. The hessian and gradient information was numerically computed by the algorithm. The optimization was carried out in a Xeon dual processor of 2.66 GHz each with 2.00 GB of RAM. The optimization frequency was chosen to be every 4 h, which was larger than the residence time of the distillation column, as the objective is to optimize the slow-scale dynamics of the overall plant. The output of the plant was corrupted with a white measurement noise with maximum magnitude of 5% of the nominal values of the output. An extended Kalman filter (EKF) was used to update the state from the measurement information, and it had a sample time of 0.02 h. The objective function was formulated as follows

$$\min_{\Delta\mathbf{u}(k),\ldots,\Delta\mathbf{u}(k+m-1)} \sum_{i=1}^{P} Q_y\left(\frac{x_{2B}^{sp}-x_{2B}(k+i)}{x_{2B,ss}}\right)^2 + \sum_{j=0}^{m-1}\left[Q_u\left(\frac{B^{sp}-B(k+j)}{B_{ss}}\right)^2 + \Delta\mathbf{u}(k+j)^T R\Delta\mathbf{u}(k+j)\right] \quad (1)$$

where $\Delta\mathbf{u} = [\Delta F_0,\ \Delta M_R^{sp},\ \Delta M_D^{sp},\ \Delta M_B^{sp},\ \Delta L,\ \Delta B]^T/100$. The prediction horizon was chosen as $p = 12$, which would cover almost the entire transient dynamics of the system. The control horizon was limited to $m = 3$, since larger control horizon demanded much more computational time. B_{ss} and $x_{2B,ss}$ are the steady-state values of B and x_{2B}, respectively. The weighting factors Q_u, Q_y, and R in Eq. 1 are 10,000, 6,000, and $20I_{6\times6}$, respectively.

Grade transition by a full-order NMPC

First, the full-order nonlinear process model is used to generate the output prediction, based on which the optimal input profile for the grade transition is computed. The result of grade transition from mode 1 to mode 2 via the full-order NMPC is shown in Figure 2. In this case, the NMPC was able to steer the composition and the production rate to the target within 12 and 20 h, respectively. The performance (E)

Table 3. Operating Range of the Inputs in the Reactor-Distillation-Recycle System

	F_0	M_R^{sp}	M_D^{sp}	M_B^{sp}	L	B
Nominal value	100	110	173	181	290	100
Lower limit	60	50	113	121	60	60
Upper limit	140	170	233	241	350	140

Table 4. Output Constraints in the Reactor-Distillation-Recycle System

	x_{3B}	F	D	V
Nominal value	0.1033	1880	1780	2070
Lower limit	0	1000	1000	1200
Upper limit	0.25	2400	2400	2600

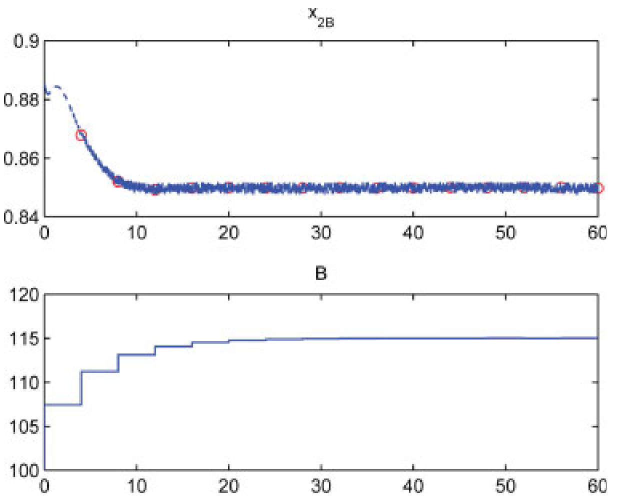


Figure 2. Product variables during the grade transition by a NMPC with full-order nonlinear model.

[Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

of the controller as measured by Eq. 2 over the 60-h period is equal to 90.70.

$$E = \sum_{k=1}^{15} \left[Q_u \left(\frac{B^{sp} - B(k)}{B_{ss}} \right)^2 + Q_y \left(\frac{x_{2B}^{sp} - x_{2B}(k)}{x_{2B,ss}} \right)^2 + \Delta\mathbf{u}(k-1)^T R \Delta\mathbf{u}(k-1) \right] \quad (2)$$

However, it required several minutes to solve the optimization of this nonlinear system as shown in Figure 3. The computational time could have become larger if the prediction horizon p was increased, such as by reducing the sample time and maintaining the 48-h prediction window. For a problem with a much larger size, this method can pose a challenge in online computation.

Grade transition by a reduced-order NMPC

Given the fact that many variables of a process system are highly correlated, it is possible and necessary to derive a significantly lower-order plant-wide model by using an appropriate method available in the literature. Most importantly, the reduced-order model should provide an accurate estimate of the slow-scale dynamics of the real plant. Nonetheless, there exists no complete theory for nonlinear model reduction,[5] unlike in the linear case. Most of the earlier works have been devoted to model simplification techniques or the singular perturbation based approaches.[6,7] While they may be well-suited to their specific problems, it becomes very complicated or impossible to generalize them to other larger system models. Therefore, we consider a projection-based proper orthogonal decomposition (POD) approach, which is a semi-empirical technique. It employs linear projection to transform the system onto the new coordinate, where a reduced-order model can be derived. This method does not require analytical derivation procedure, and therefore can be applied to high-dimensional nonlinear systems. Mathematical procedures of the POD approach can be found in Appendix A.

To identify the projection matrices, the inputs to the plant were excited in the range of $\pm 10\%$ around their nominal operating points. Eigenvalue plot of the covariance matrix from the resulting simulation data is shown in Figure 4. This suggested that only the first seven eigenvectors have significant values. Hence, the reduced-order model via residualization method contains 7 ODEs and 32 algebraic equations, while the truncation method considers only the 7 ODEs. Using the reduced-order NMPC based on residualization method to perform grade transition resulted in the output profiles shown in Figure 5. This controller can also steer the system to the mode 2 successfully. The performance over the 60-h period is 91.73. On the other hand, the NMPC based on

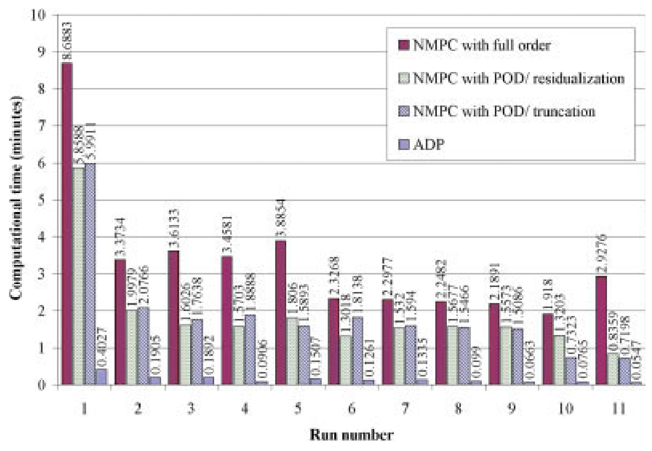


Figure 3. Computational time (in minutes) of different controllers running at every 4 h.

[Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

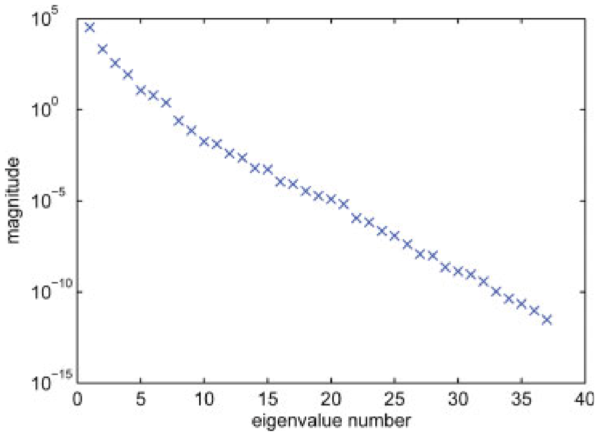


Figure 4. Eigenvalue plots of the data collected from dynamic simulations of the reactor-distillation-recycle system.

[Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

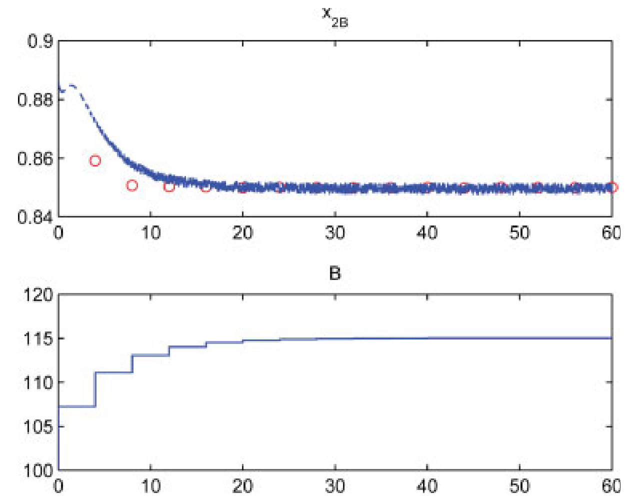


Figure 5. Product variables during the grade transition performed by a NMPC derived from a reduced-order model with residualization (solid: output, o: prediction).

[Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

the truncation method shows the worst performance of 96.02. The output trajectory from this approach is shown in Figure 6, where there is an offset between the model prediction and the actual output concentration. This is because the changes of the fast modes occur and reach the new steady states so quickly. Without taking into account these changes, the truncation approach results in the approximation error. In terms of computational time comparison, the reduced-order NMPCs require almost half the time of the full-order NMPC as shown in Figure 3. Nonetheless, when the problem size is larger, the computational time resulting from the reduced-order NMPC can still be an issue.

Approximate Dynamic Programming (ADP) Approach

Alternatively, an approximate dynamic programming (ADP) method can be applied to this problem. In this section, we first provide an overview of the dynamic programming (DP) framework and our previously published ADP method, and then discuss the computational problems associated with it. We propose systematic methods to reduce the offline computational load so that the ADP method can be effectively used to control a high-dimensional integrated plant of the case study.

Overview of dynamic programming framework

The problem of finding an optimal state feedback policy for a deterministic state-space system may be represented as follows

$$\min_{\mu\in\Pi}\sum_{i=0}^{\infty}\phi(x(i),u(i)) \tag{3}$$

subject to

$$\begin{aligned}&\text{FeedbackPolicy:}\ u(i)=\mu(x(i))\\&\text{Pathconstraint:}\quad g(x(i),u(i))\geq 0,\qquad i=0,1,\ldots,\ldots\\&\text{Modelconstraint:}\ x(i+1)=F_h(x(i),u(i))\equiv\int_{i\Delta t}^{(i+1)\Delta t}f(x(\tau),u(\tau))d\tau\end{aligned} \tag{4}$$

where $x(i)$ represents a state vector at the i^{th} sample time, u a vector of manipulated variables, and Δt the sampling interval. μ is a state feedback policy, and Π is the set of all admissible policies over which optimal μ is to be found. Here we do not put any restriction other than that it should be a function mapping $x(i)$ to $u(i)$. The stage-wise cost function is denoted by ϕ. The path and model constraints may be nonlinear functions. A continuous process model $\dot{x}=f(x,u)$ is assumed to be available to be integrated over the sample time Δt with a piece-wise constant input $u(\tau)$ = $u(i)$ for $i\,\Delta t \leq \tau \leq (i+1)\Delta t$.

The DP framework is based on the *Bellman's optimality principle*, which is used to derive the optimal control policy. In this approach, we define the "cost-to-go" of a starting state x under the control policy μ, denoted by $J^{\mu}(x)$, as the sum of a stage-wise cost incurred from the state x over the infinite horizon, i.e.,

$$J^{\mu}(x)=\sum_{i=0}^{\infty}\phi(x(i),\mu(x(i))),\quad x(0)=x \tag{5}$$

$J^{\mu}(x)$ is assumed to be well-defined over the entire S, which is a set of all possible state points. To address states with an unbounded cost-to-go value, one can saturate the function at a very large value (relative to the cost-to-go values of other states). The goal of DP is to derive the optimal cost-to-go function J^*, which is the cost-to-go function under the optimal policy. That is, $J^*=\min_{\mu}J^{\mu}$.

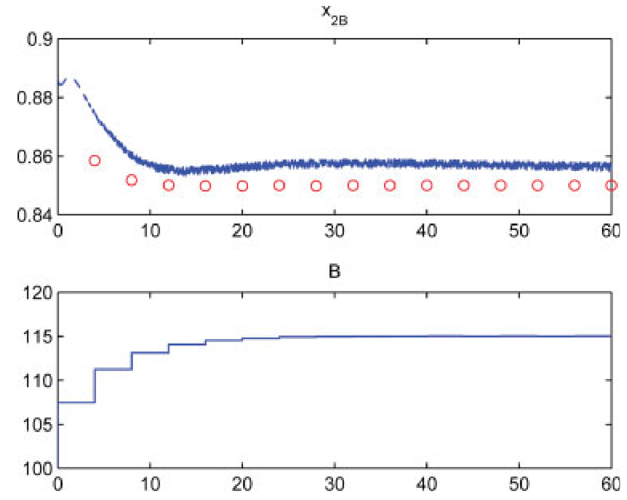


Figure 6. Product variables during the grade transition performed by a NMPC derived from a reduced-order model with truncation (solid: output, o: prediction).

[Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

From Bellman's optimality principle, it follows that the optimal cost-to-go function satisfies the following *Bellman's optimality equation*[8]

$$J^*(x(i)) = \min_{u \in U}[\phi(x(i), u(i)) + J^*(x(i+1))], \quad \forall x \in S \tag{6}$$

where U is a set of all possible actions. The objective of the offline cost-to-go learning is to arrive at the optimal cost-to-go function. Once J^* value for every state point is known, the solution to an online optimization problem (3) can be obtained by solving an equivalent single-stage optimization problem shown below. This can be done offline creating a policy lookup table, or online for a specific state encountered at each time. Even in the latter case, the online computational load can be drastically reduced

$$\mu^*(x(i)) = \arg\min_{u(i) \in U}[\phi(x(i), u(i)) + J^*(F_h(x(i), u(i)))] \tag{7}$$

For stochastic systems, the optimization problem can be represented as follows

$$\min_{\mu \in \Pi} \mathbf{E}\left[\sum_{i=0}^{\infty} \alpha^i \phi(x(i), u(i))\right] \tag{8}$$

$$u(i) = \mu(I(i)) \tag{9}$$

where $\mathbf{E}$ is an expectation operator. $\alpha \in [0,1)$ is a discount factor that signifies the importance of the immediate cost compared to the future cost. Use of $\alpha < 1$ is needed in stochastic problems in order that the infinite horizon cost function is bounded. In this case, the control policy determines the action based on the available stochastic information vector denoted by I. This information vector typically includes the conditional probability distribution of the state vector.

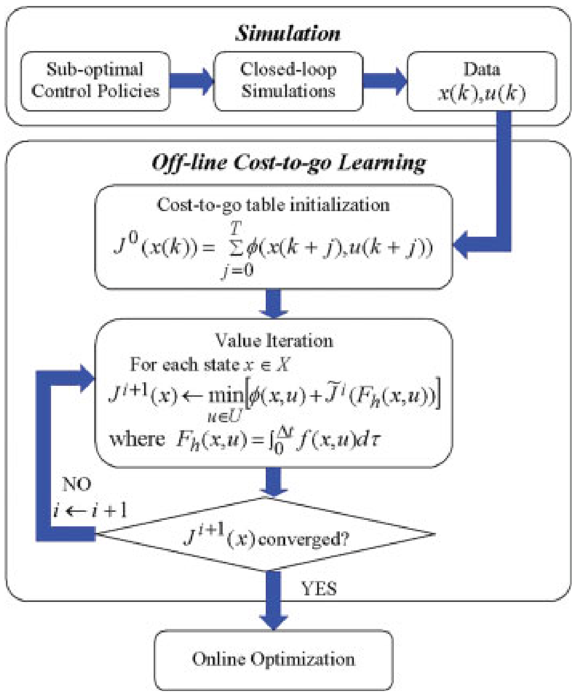
The Bellman equation for stochastic system is defined as the following

$$J^*(I(i)) = \min_{u(i)} \mathbf{E}[\phi(x(i), u(i)) + \alpha J^*(I(i+1))|I(i)] \tag{10}$$

where it is assumed that the stochastic equation governing the one-time step transition of I is available. Since the solution to the Bellman equation can seldom be obtained in a closed form, value iteration is one of the most widely used algorithms to obtain the optimal cost-to-go function numerically. For a discrete finite state space problem, value iteration starts with an initial estimate of the cost-to-go for each state, and then iterates the Bellman equation for every state point by considering every possible action until convergence. In each iteration, every state point is updated once in a sequential manner. This can lead to a prohibitively large offline storage and computational load as the size of the state and action space becomes larger. This problem is known as the "*curse of dimensionality*," which makes DP impractical for most practical size problems.

Overview of the existing ADP method

To overcome the curse-of-dimensionality problem, an approximate dynamic programming (ADP) framework has been widely studied in the literature in order to approximate the optimal cost-to-go function.[9,10] Among these works, only a few can be applied to process control problems in which the state space and the action space are continuous and discretization leads to a very large number of state and action points to explore. Training data typically span only a small portion of the state space, because chemical processes are normally controlled within small operating regions. Earlier approaches applied to continuous state space problems used global approximators, such as artificial neural networks in the neurodynamic programming (NDP) framework,[11] to fit simulation data of the state vs. future cost value to the cost-to-go function. When this technique is applied to process control problems with sparse training data, divergence often results during the iterative offline learning due to over-fitting.[2] To overcome such problems, Lee and coworkers[2] combined the lookup-table approach with the use of a local approximator with nonexpansion property, particularly a k-nearest neighbor averager, and a quadratic penalty term to guard against over-extrapolation of the registered cost-to-go data into regions not covered by the training data. Their suggested ADP framework is depicted in Figure 7. The key idea of this framework is not to solve for the optimal cost-to-go function for the entire state space, but to use the closed-loop simulations with suboptimal control policies to generate state trajectories that cover the state space, denoted by X, relevant to the optimal control problem. The premise is that optimally controlled trajectories are confined to a small set of the state space. Therefore, the search for an optimal policy could be limited to the "relevant" state space. However, this relevant space is not known *a priori*, and, hence, the next best option is to approximate it by simulating a number of suboptimal policies under all potential operating and disturbance conditions. Once the simulation data is available, the cost-to-go



Simulation
Sub-optimal Control Policies
Closed-loop Simulations
Data $x(k), u(k)$
Off-line Cost-to-go Learning
Cost-to-go table initialization
$J^0(x(k)) = \sum_{j=0}^{T} \phi(x(k+j), u(k+j))$
Value Iteration
For each state $x \in X$
$J^{i+1}(x) \leftarrow \min_{u \in U}[\phi(x,u) + \tilde{J}^i(F_h(x,u))]$
where $F_h(x,u) = \int_0^{\Delta t} f(x,u)d\tau$
NO
$i \leftarrow i+1$
$J^{i+1}(x)$ converged?
YES
Online Optimization

Figure 7. Algorithmic framework of approximate dynamic programming.

[Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

lookup table can be initialized by summing the cost starting from each state point until the error becomes approximately zero. Then, the cost values in the lookup table are iteratively improved in the offline value iteration step.

Cost-to-go function approximation

In the algorithm proposed by Lee et al.,[2] the cost-to-go update of each state point is done by evaluating every discretized action and computing the corresponding immediate cost and the cost-to-go of the successor state, as shown in Eq. 11 and Figure 7

$$J^{i+1}(x) \leftarrow \min_{u \in U} \phi(x,u) + \tilde{J}^i(F_h(x,u)) \tag{11}$$

Since the subsequent state $F_h(x, u)$ may not be prestored in the lookup table, the cost-to-go approximation scheme has to be used. As mentioned previously, because the effective operating space of a process control problem can be much smaller than the entire state space, use of global approximators can often lead to convergence problems during the value iteration. Therefore, the use of k-nearest neighbor averager for cost-to-go approximation, and the parzen density estimation to guard against extrapolation are more suitable for process control problems.[2] In this approach, there are two steps in computing the cost-to-go estimate:

1. Local cost-to-go approximation: The k-nearest neighbor approach approximates the cost-to-go of the query point $x_0 = F_h(x, u)$ as a weighted average of the cost-to-go values of its closest neighbors. The first step is to compute the Euclidean distance between the query point x_0 and every state point x_i in X, where X is a set of the stored state points as follow

$$d_i = \sqrt{(x_0 - x_i)^T W (x_0 - x_i)} \tag{12}$$

where W is a user-defined diagonal matrix assigning weights to different state variables. Then the distance is sorted and only the k closest points are considered for the cost-to-go approximation

$$\tilde{J}(x_0) = \sum_{x_i \in N_k(x_0)} w_i J(x_i) \tag{13}$$

$$w_i = \frac{1/d_i}{\sum_{i=1}^{k} (1/d_i)} \tag{14}$$

where $N_k(x_0)$ is a set of the k-nearest neighbors of x_0 and w_i is a distance weighting parameter.

2. Local data density estimation: To discourage excessive extrapolation into the region far away from where training data occupy, Parzen data density estimator is employed. The Parzen density estimate of a query point x_0 in the training data set X is defined as

$$f_X(x_0) = \frac{1}{N\sigma^n} \sum_{i=1}^{N} K\left(\frac{x_0 - x_i}{\sigma}\right) \tag{15}$$

where n is the state dimension, x_i represents the stored state point, N is the number of stored state points in X, σ is a parameter defining confidence bandwidth, and K is a multivariate Gaussian Kernel function. Hence, the parzen density function is a measure of "closeness" of store data points based on the Euclidean distance between the query data point x_0, and every point in the stored data set. Kernel function value is high when the data points are close to the query point, and low when they are far away. Hence, if the Parzen density estimate of the query point is too small, a quadratic penalty term is added to the cost-to-go estimate for the query point as follows

$$J_{\text{bias}}(x_0) = \begin{cases} J_{\max}\left[\frac{(f_X(x_0))^{-1} - \rho}{\rho}\right]^2 & \text{if} (f_X(x_0))^{-1} > \rho \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

$$\tilde{J}(x_0) \leftarrow \tilde{J}(x_0) + J_{\text{bias}}(x_0) \tag{17}$$

where parameters $J_{\max}$ is the upper bound on the cost-to-go value, and ρ is the threshold value. A guideline for designing these parameters can be found in Lee et al.[2]

Computational issues of the existing ADP method

According to the procedure of the ADP approach described previously, total offline value iteration time of the ADP approach is proportional to (1) the number of discretized actions in the action space, (2) the time required to approximate the cost-to-go value of the successor state resulting from each action in Eq. 11, (3) the number of stored state points N, and (4) the number of iterations needed for the cost-to-go function to converge. This article is concerned with reducing the computational time arising from the first three factors. The number of iterations required typically depends on the effectiveness of the initial control policy used in the simulation. The closer they are to the optimal policy, the less number of iterations is required in general.

Curse of dimensionality in the action space

In this case study, the action space has six dimensions with the operating range shown in Table 3. Clearly, there need to be many discretized values for each input variable in order to ensure good solution quality. Unfortunately, this will lead to a heavy computational requirement. Even if we use a coarsely discretized grid, there will still be too many action combinations to evaluate in order to update the cost-to-go of each state point. For example, if the grid size is 5 for each input, there will be as many as $16 \times 24 \times 24 \times 24 \times 58 \times 16 \sim 205 \times 10^6$ action combinations! Clearly, the existing ADP procedure suggested by Lee et al[2] can lead to the curse-of-dimensionality in the action space for problems with many action dimensions.

Computational load to approximate $\sim \tilde{J}(F_h)$ per action

As explained in the previous section, approximating the cost-to-go value of the successor state for each action u involves the model integration and the computation of Eqs. 12–17. We perform the value iteration on a Xeon™ dual processor with 2.66 GHz/2.66 GHz and 2.00 GB of RAM, and using MATLAB 7 software. The average time to compute an approximation of the cost-to-go in Eq. 11 per state per action was 0.664 s. Suppose that there are 1,000 stored data points. If we assume optimistically that the optimal solution to Eq. 11 can be found within some heuristic trials of 1,000 actions, then the time required to perform a single round of value iteration would be $\sim$ 0.664 s $\times$ 1,000 $\times$ 1,000 = 7.7 days! If the starting control policies used for simulation were far from optimal, number of iterations for the cost-to-go to converge could be large. This means that it could take several months to have the converged cost-to-go function for online optimization!

Proposed methodologies to apply ADP to optimal control of an integrated plant

In this section, we propose methodologies to apply ADP to optimal control of an integrated plant. First, the computational load associated with a large action space is to be reduced. In addition, for problems with high-dimensional state space, the feature weighting matrix in the *k*-nearest neighbor cost-to-go approximating step has to be carefully chosen so that the approximation is accurate. Furthermore, removing state dimensions irrelevant to the system's dynamic evolution from the computation of Eq. 12 will reduce the computational overhead. Finally, because the cost-to-go update requires a simulation to generate the state transition, the reduction in model integration time contributes to further reduction of the overall offline computational time.

Nonlinear optimization based search approach for finding optimal action

As mentioned in the previous section, applying an exhaustive search strategy for optimal action to high-dimensional problems can lead to an extremely heavy computational load. An alternative is to systematically direct the search in a way that allows quick convergence to locally optimal action values. The approach suggested here is to apply a nonlinear optimization method for finding an optimal action. We can further reduce the computational load by restricting the search region for the optimal action in order that the optimizer converges quickly.

Because the dynamical system and the cost-to-go function studied in this work are nonlinear, an efficient nonlinear optimization algorithm has to be used. This can be a general-purpose NLP solver, such as the sequential quadratic programming (SQP) solver. A more sophisticated simultaneous optimization solver can be applied, although it is not clear whether it will give an advantage over the sequential optimization approach. This is because in this case integration of the model equation for only one sample interval is required, which greatly simplifies the implementation of the sequential optimization.

In this case the cost-to-go update for each state point $x \in X$ is the solution from the following constrained optimization

$$J^{i+1}(x) = \min \phi(x,u) + \tilde{J}^i(F_h(x,u)) \tag{18}$$

$$\text{s.t.} \quad x_{LB} \le F_h(x,u) \le x_{UB}$$
$$u_{LB} \le u \le u_{UB}$$

where the subscripts *LB* and *UB* denote the lower and the upper bounds, respectively. Output constraints and a good initial guess are given by the following *local action set* heuristic.

Local Action Set Heuristic. Our proposed heuristic is to construct for each state point $x \in X$ a local action set denoted by A^x. This local action set is initialized by storing good actions that have been applied to the state point x, and the nearest neighbors of x during the offline simulation. Then, the cost-to-go update of each state point x is done by solving the optimization problem (Eq. 18), with the input feasible region be the polyhedron spanned by the local action set A^x. That is, u_{LB} and u_{UB} are constructed from the minimum and maximum magnitudes in each input dimension of A^x. By including the information of past local actions, we can direct the search to the region where good control actions can be found.

Dimensionality reduction in computing k-nearest neighbor

As described in the previous section, this ADP algorithm requires the computation of distance between the query point and every stored state point. Oftentimes for processes with large state dimensions, not every state variable is relevant for determining the cost-to-go value of the state. Therefore, those state dimensions that do not affect the cost-to-go value can be taken out from the Euclidean distance computation (d_i) to save the computational overhead and to improve the prediction accuracy of the *k*-nearest neighbor cost-to-go approximation. Nonetheless, the naive approach that keeps only the variables that show up explicitly in the stage-wise cost function may leave out other state variables that are strongly correlated with the cost-to-go. Therefore, we suggest the use of a systematic *variable selection* or *feature selection* technique to select a subset of state variables from the original state vector and use them in the *k*-nearest neighbor cost-to-go value approximation. There are several techniques in the literature that attempt to capture the correlation between the predictors and the regressors, including linear projection techniques like the partial least-square (PLS) method.[12] Choice of methods is problem-dependent and users should cross-validate the chosen method to make sure the reduced-dimensional state vector still captures the relationship between the state and the cost-to-go value adequately. In addition, regression coefficients between the state variables and the cost-to-go value can be used as the feature weights in the Euclidean distance computation (Eq. 12). Let the superscript r denote the state vector after removing the variables with small coefficients, and B^r be a diagonal matrix containing only large coefficients of the remaining states. The distance computation in Eq. 12 can be replaced by

$$d_i = \sqrt{(x_0^r - x_i^r)^T B^r (x_0^r - x_i^r)}.$$

Reducing model simulation time via nonlinear model reduction

Integrating high-order nonlinear dynamic plant model to find the successor state can be computationally demanding

Table 5. Modified ADP Algorithm with Local Gradient-Based Search for Optimal Action

```
1. Close-loop simulation with known control policies
   $\mu_i, i = 1, \ldots, n_\mu$. Store visited states $\{x(1), \ldots, x(N)\} \equiv X$ and
   the actions applied to the state point
   $x(j) : u^{x_j} = \mu_i(x(j)), j = 1, \ldots, N$.
2. Initialization of the cost-to-go table:
   $J^0(x(k)) = \sum_{j=0}^{T} \phi(x(k+j), u(k+j)),\ k = 1, \ldots, N$, where $T$ is
   sufficiently large for the remaining stage-wise cost to be
   negligible.
3. Initialization of the local action sets.
   For each $x$, construct a local action set $A^x$ from
   $[u^x, u^{x_1}, \ldots, u^{x_m}] \in A^x$, where $x_j, j = 1, \ldots, m$ is the first $m$ nearest
   neighbor points of $x$.
4. Data Analysis for dimensionality reduction:
   - Apply a feature selection method (such as the PLS method) to
   find appropriate feature weighting matrix and to reduce the
   dimension in the $k$-nearest neighbor procedure.
   - Obtain a reduced-order model (such as via POD/residualization
   technique).
5. Value Iteration Procedure:
   REPEAT
     FOR each $x \in X$, solve
       $J^{i+1}(x) = \min \phi(x, u) + \tilde{J}^i(F_h(x, u))$
       s.t.  $x_{LB} \le F_h(x, u) \le x_{UB}$
         $u_{LB} \le\ u\ \le u_{UB},\ u_{LB} = \min(A^x),\ u_{UB} = \max(A^x)$
       where $\tilde{J}^i$ is the $k$-nearest neighbor cost-to-go approximation
     LOOP
   $i \leftarrow i + 1$
   UNTIL convergence
```

especially when the system is "stiff," which is often the case when the plant has a large recycle loop, as in this example. Given the fact that many variables of a plant-wide system are often highly correlated, it is possible to derive a significantly lower-order plant model using an appropriate method that preserves the accuracy of the resulting model. In this work, we consider the proper orthogonal decomposition (POD) method coupled with the residualization technique presented in the previous section as a step to reduce the model integration time.

With all the schemes proposed in the previous three sections, the modified ADP procedures can be summarized in Table 5.

Results of ADP applied to a deterministic nonlinear optimal control problem

In this section, we implemented the proposed ADP method to the grade transition problem of an integrated plant shown in Figure 1. To generate the sample trajectories of the grade transitions from mode 1 to 2, 3, and 4, we used a reduced-order NMPC controller designed with prediction and control horizons of 5 and 2, respectively. We varied the weighting parameters of the controller so as to cover wider regions of the state space. The sample time of the controller was chosen as 4 h. Here we assumed that the full state variables were measured. Total number of sample data collected was 990 points. The state for ADP has to be defined in a way that can adequately distinguish different data points. In this case, the state was defined as follows

$$X = \left[x_1^{\text{plant}}, \ldots, x_{39}^{\text{plant}}, x_{3B}, F, D, V, (x_{2B}^{sp} - x_{2B}), (B^{sp} - B), x_{2B}^{sp}, B^{sp}\right] \quad (19)$$

where all the output variables of the process are included. In addition x_{3B}, F, D, and V were also included, since they are constrained variables for the optimal control problem. The set points, as well as errors from set points were part of the state as they are required to compute the stage-wise cost, which is defined in Eq. 20, where $Q_u = 10{,}000$, $Q_y = 6{,}000$, and $R = 20I_{6\times 6}$

$$\phi(x(k), u(k)) = Q_y \left(\frac{x_{2B}^{sp} - x_{2B}(k+1)}{x_{2B,ss}}\right)^2 + Q_u \left(\frac{B^{sp} - B(k)}{B_{ss}}\right)^2 + \Delta\mathbf{u}(k)^T R \Delta\mathbf{u}(k) \quad (20)$$

In this work, we design the confidence bandwidth parameter σ as 3% of $6\sqrt{n}$, where n is the state dimension. The initial cost-to-go values of the stored state points range from 0 to 479. A quadratic penalty term is added to discourage extrapolation to the regions not covered by the simulation data. The maximum cost-to-go value $J_{\max}$ is set to 1,000. The number of nearest neighbors used for the cost-to-go estimation was 4.

Implementation of the improved offline learning approach

First, the curse of dimensionality in the action space was overcome by using the SQP technique to find the optimal action within the region spanned by past actions stored for a given state and its neighbors. For each sample point, the number of nearest neighbors whose actions were kept in the local action set was 10. Since during our offline simulation we had applied good MPC controllers, actions applied to the nearest neighbors provided good starting points from which the SQP search could be performed. In most cases, the search converged quickly to the locally optimal solution. Had the initial controllers been very poor, one should consider exploring many more neighbors' actions to expand the search region.

To reduce the state dimension in the k-nearest neighbor computation, the PLS technique was applied to determine the correlations between the state and the cost-to-go value. The PLS model showed 13 states with large coefficients as shown in Table 6. As a result, only these state variables were used in the k-nearest neighbor computation and their correspond-

Table 6. Descriptions of the State Variables with Large PLS Regression Coefficients

State Number	Description	Coefficient
1	Reactor holdup (M_R)	0.668
2	Composition of A in the reactor (x_{1R})	0.474
3	Composition of B in the reactor (x_{2R})	0.556
33	Composition of A on tray 14 ($x_{1,14}$)	0.287
35	Composition of A on tray 15 ($x_{1,15}$)	0.394
36	Composition of B on tray 15 ($x_{2,15}$)	0.686
38	Composition of A in the product (x_{1B})	0.409
39	Composition of B in the product (x_{2B})	0.563
40	Impurity in the product (x_{3B})	−0.567
41	Reactor effluent flow rate (F)	−0.730
45	Error from x_{2B} setpoint (B^{sp} – B)	0.290
46	Error from B setpoint (x_{2B}^{sp})	0.284
47	Production rate setpoint (B)	−0.678

Table 7. Value Iteration Time of Different Improvement Methods

Methods	Average computational time per iteration (minutes)
SQP search over local action sets	595.13
SQP search over local action sets + reduced order model for state transition	112.30
SQP search over local action sets + reduced order model for state transition + reduced dimension in k-nearest neighbor computation	92.31

ing feature weights were set as the magnitudes of the PLS regression coefficients.

For the nonlinear model integration to find a successor state, we applied the POD method to reduce the model integration time. The value iteration step performed with the convergence criteria $\left|J^{i+1}(x) - J^{i}(x)\right|_{\infty} < 1$ converged after 18 iterations. The offline computational time before and after the improvement methods were applied are summarized in Table 7. Clearly, the SQP search over local action sets allowed the action space to be explored without the curse of dimensionality and resulted in an average computational time of 10 h per iteration. Then, combined with the model reduction technique, the offline computational time was further reduced to 2 h per iteration. Finally, reducing the state dimension in the k-nearest neighbor computation allowed the cost-to-go to be updated much more efficiently within 1.5 h per iteration.

Online performance comparison

Once the cost-to-go table converged; it was used in the online optimization. The result of grade transition from mode

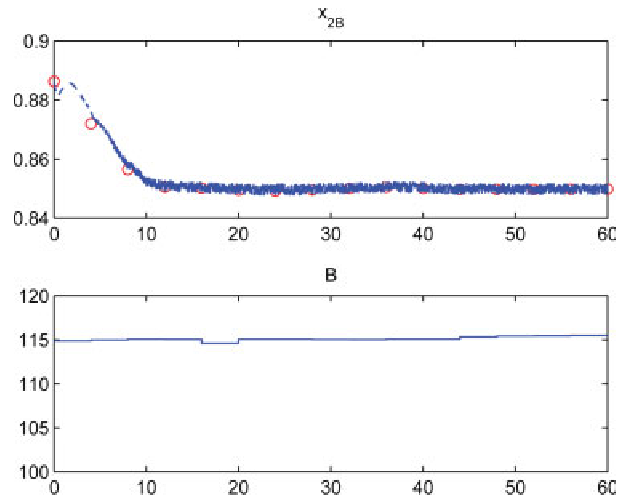


Figure 8. Product variables during the grade transition performed by ADP controller (solid: output, o: prediction).

[Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

Table 8. Online Performance Comparison Between of NMPC and ADP Schemes Running at Every 4 hours

Method	Performance
NMPC with a full-order model	90.70
NMPC with a reduce-order model via residualization	91.73
NMPC with a reduced-order model via truncation	96.02
ADP-based controller	13.27

1 to 2 in Figure 8 shows that the system can achieve the grade transition very quickly. The resulting performance as defined by Eq. 2 is 13.27, which represents a significant improvement from the use of the NMPC controllers as compared in Table 8. This can be attributed to the fact that the problem ADP solves is an infinite horizon problem, whereas the NMPC's prediction and control horizons were limited due to the computational reason. In addition, the main benefit of the ADP method is in its drastic reduction in the online computation as shown in Figure 3. This is due to the fact that the ADP controller only needs to solve a single-stage optimization problem at each time, which is a much smaller problem than those solved in the NMPC.

Results of ADP application to a stochastic nonlinear optimal control problem

In this section, we consider the case where integrated white noises are introduced to the feed composition and the kinetic rate constant k_1. The feed now consists of components A (with composition of $x_{1,0}$) and C (with composition of $x_{3,0}$). The changes in $x_{3,0}$ and k_1 have the sample time of 0.02, and are modeled as integrated white noise.

To generate representative coverage of the disturbance space, while maintaining a reasonable size of the data set, we chose the following realizations of the stochastic disturbances for the simulation: (1) k_1 and $x_{3,0}$ are trending up, (2) k_1 is trending down and $x_{3,0}$ is trending up, (3) k_1 is trending up and $x_{3,0}$ is up for the first 30 h and then down, and (4) k_1 is trending down, and $x_{3,0}$ is up for the first 30 h and down from there.

We designed seven reduced-order NMPC controllers with the sample time of 4 h for use as the initial controllers for simulations. The prediction and the control horizons of all controllers were chosen as 5 and 2, respectively. The weighting matrices of these controllers are shown in Table 9. We consider the grade transition scenario from mode 1 to mode 2. We store 600 representative data points from the simulations for the cost-to-go training.

The discount factor used to compute the cost-to-go is 0.9. The cost-to-go update in the offline value iteration was per-

Table 9. Weighting Matrices of the Initial Controllers

	Controller Number						
Parameter	1	2	3	4	5	6	7
Q_u	10000	10000	1250	10000	80000	10000	10000
Q_y	6000	96000	6000	6000	6000	750	3000
R	$20I_{6\times6}$	$20I_{6\times6}$	$20I_{6\times6}$	$160I_{6\times6}$	$20I_{6\times6}$	$20I_{6\times6}$	$160I_{6\times6}$

Table 10. Online Cost Comparison with 12 Stochastic Disturbance Scenarios

Controller	ADP	NMPC 1	NMPC 2	NMPC 3	NMPC 4	NMPC 5	NMPC 6	NMPC 7
mean	37.83	130.28	122.33	136.56	152.98	125.52	147.43	131.21
S.D.	5.30	22.91	13.62	18.26	27.84	17.55	29.76	20.70

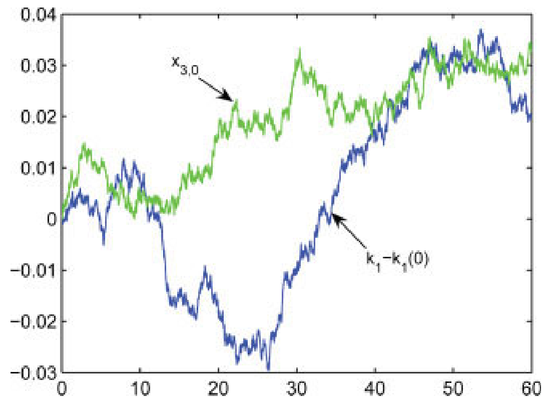


Figure 9. Disturbance scenario 6.

[Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

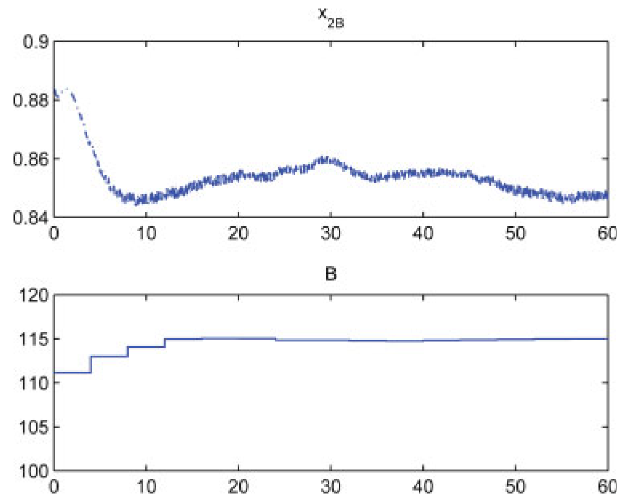


Figure 10. Online performance of the ADP controller during the disturbance scenario 6.

[Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

formed by taking the expectation of the cost over 30 stochastic disturbance realizations. The initial cost-to-go values of the stored state points ranged between 0.398 and 167. The convergence criteria of the iteration was chosen as $\left|J^{i+1}(x) - J^i(x)\right|_\infty < 1$, which is small compared to the range that the cost-to-go values varied. The cost-to-go learning converged after 12 iterations. The online performance was compared by generating 12 new realizations of the stochastic disturbances in k_1 and $x_{3,0}$. We compared the performance of the ADP derived controller with those of the seven reduced-order NMPCs used as the initial controllers for simulations.

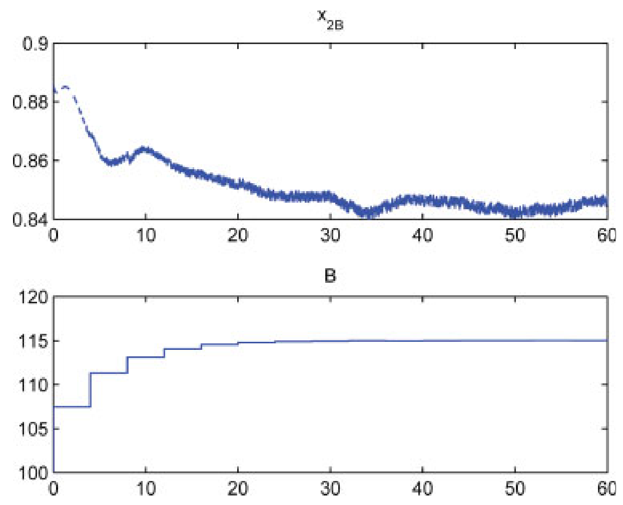


Figure 11. Online performance of the NMPC controller 1 during the disturbance scenario 6.

[Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

The performance as measured by Eq. 2 over 60-h horizon is shown in Table 10. The ADP based controller shows a much lower mean cost and standard deviation compared to those of all the other reduced-order NMPC controllers.

For illustration, the disturbance scenario 6 was the one that the ADP controller showed the median performance of 34.27. The disturbances are plotted in Figure 9. The performance of the ADP controller is shown in Figure 10. From the time 10 h, the production concentration was closely controlled around 0.85–0.86, while the production rate reached the new target in 12 h. The performance of NMPC number 1, which was the best performing controller among all the NMPC controllers in this case, is shown in Figure 11. From the time 10 h, the product concentration varied between 0.84–0.86. In this case, the production rate reached the target after 20 h.

Conclusions

The approximate dynamic programming (ADP) approach proved to be an efficient method for online optimization of an integrated plant offering significant reduction in the online computation and improved performance in the presence of stochastic disturbances. However, the existing offline learning procedures can take an unreasonable amount of time for the cost-to-go value function to converge. Therefore, this case study presented a set of methods to reduce the search space for optimal action and the time taken in the cost-to-go update of each state point. We incorporated nonlinear programming based search technique to accelerate the convergence to a local optimal action. The reduction of state dimension via the partial least-square (PLS) technique was shown to be effective in reducing the computation time for searching nearest neighbors and computing a cost-to-go estimate. Finally, the nonlinear model reduction technique via proper orthogonal decomposition (POD) combined with residualization was applied to reduce the model integration time. With these improvement techniques, the ADP method has successfully been applied to solve an integrated plant control problem, resulting in superior performance compared to the conventional NMPC method.

Literature Cited

1. Lee JM. *A study on architecture, algorithms, and applications of approximate dynamic programming based approach to optimal control*. Georgia Institute of Technology; 2004. Ph.D. Thesis.
2. Lee JM, Kaisare NS, Lee JH. Choice of approximator and design of penalty function for an approximate dynamic programming based control approach. *J Process Control*. 2006;16:135–156.
3. Kumar A, Daoutidis P. Nonlinear dynamics and control of process system with recycle. *J Process Control*. 2002;12:475–484.
4. Kaisare NS, Lee JM, Lee JH. Simulation based strategy for nonlinear optimal control: Application to a microbial cell reactor. *Int J Robust Nonlinear Control*. 2003;13:347–363.
5. Marquardt W. Nonlinear model reduction for optimization based control of transient chemical processes. In: *Proceedings of the 6th International Conference on Chemical Process Control*. Tucson, AZ; 2001:30–60.
6. Hahn J, Edgar TF. An improved method for nonlinear model reduction using balancing of empirical gramians. *Comput Chem Eng*. 2002;26:1379–1397.
7. van den Berg J. *Model reduction for dynamic real-time optimization of chemical processes*. Delft University of Technology, The Netherlands; 2005. Ph.D. Thesis.
8. Bellman RE. *Dynamic Programming*, New York: Dover Publications, Inc., 2003.
9. Lee JM, Lee JH. Approximate dynamic programming strategies and their applicability for process control: A review and future directions. *Int J Control, Automation, and Syst*. 2004;2:263–278.
10. Si J, Barto AG, Powell WB, Wunsch II D. *Handbook of Learning and Approximate Dynamic Programming*. New Jersey: IEEE Press; 2004.
11. Bertsekas DP, Tsitsiklis J. *Neuro-Dynamic Programming*. New Hampshire: Athena Scientific; 1996.
12. Wold S, Ruhe A, Wold H, Dunn IIIWJ. The colinearity problem in linear regression. The partial least square approach to generalized inverses. *SIAM J Sci Comput*. 1994;5:735–743.

Appendix A: Proper Orthogonal Decomposition (POD) Method

For a nonlinear ODE system of the form

$$\dot{x} = f(x,u), \quad x \in \Re^n \tag{A1}$$

the model reduction technique based on POD attempts to find the projection matrices P and Q that transform the system to a new coordinate consisting of slow modes ($\bar{x}_1$) and fast modes ($\bar{x}_2$) according to:

$$\bar{x} = \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \end{bmatrix} = \begin{bmatrix} P \\ Q \end{bmatrix}(x - x_{ss}) \tag{A2}$$

To find these projection matrices, simulation data of the dynamical system are collected. Then, eigenvectors of the covariance matrix of the data are calculated. The row vectors in the projection matrix P are selected to be the first r eigenvectors, whose eigenvalues are larger than zero. The rest of the eigenvectors are the row vectors in the matrix Q. Once the order r is chosen, the original nonlinear system (Eq. A1) can be transformed to

$$\dot{\bar{x}}_1(t) = Pf([P^T, Q^T]\bar{x} + x_{ss}, u(t)) \tag{A3}$$

$$\dot{\bar{x}}_2(t) = Qf([P^T, Q^T]\bar{x} + x_{ss}, u(t)) \tag{A4}$$

Then, the reduced-order dynamic model can be derived by two approaches.

Residualization. This method assumed that the fast modes are at the quasi-steady state. Therefore, the resulting reduced-order system is in the following DAE form

$$\begin{aligned} \dot{\bar{x}}_1(t) &= Pf([P^T, Q^T]\bar{x} + x_{ss}, u(t)) \\ 0 &= Qf([P^T, Q^T]\bar{x} + x_{ss}, u(t)) \end{aligned} \tag{A5}$$

Truncation. On the other hand, the truncation method assumed that the fast modes do not change at all. Therefore, only the state $\bar{x}_1$ is solved in the reduced-order ODE system of the form

$$\dot{\bar{x}}_1(t) = Pf(P^T\bar{x}_1 + x_{ss}, u(t)) \tag{A6}$$

Manuscript received Feb. 22, 2008, and revision received Sept. 7, 2008, and final revision received Dec. 12, 2008.