Available online at www.sciencedirect.com
SCIENCE DIRECT®
automatica
Automatica 41 (2005) 1281–1288
www.elsevier.com/locate/automatica

Brief paper

Approximate dynamic programming-based approaches for input–output data-driven control of nonlinear processes☆

Jong Min Lee, Jay H. Lee*

311 Ferst Dr. NW, School of Chemical and Biomolecular Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0100, USA

Received 10 May 2004; received in revised form 30 December 2004; accepted 20 February 2005
Available online 15 April 2005

Abstract

We propose two *approximate dynamic programming* (ADP)-based strategies for control of nonlinear processes using input–output data. In the first strategy, which we term 'J-learning,' one builds an empirical nonlinear model using closed-loop test data and performs dynamic programming with it to derive an improved control policy. In the second strategy, called 'Q-learning,' one tries to learn an improved control policy in a model-less manner. Compared to the conventional model predictive control approach, the new approach offers some practical advantages in using nonlinear empirical models for process control. Besides the potential reduction in the on-line computational burden, it offers a convenient way to control the degree of model extrapolation in the calculation of optimal control moves. One major difficulty associated with using an empirical model within the multi-step predictive control setting is that the model can be excessively extrapolated into regions of the state space where identification data were scarce or nonexistent, leading to performances far worse than predicted by the model. Within the proposed ADP-based strategies, this problem is handled by imposing a penalty term designed on the basis of local data distribution. A CSTR example is provided to illustrate the proposed approaches.

Keywords: Nonlinear model identification; Nonlinear model predictive control; Approximate dynamic programming; NARX model; Reinforcement learning; Q-learning

1. Introduction

Whereas the dynamic behavior of most chemical processes is nonlinear, linear models have predominantly been used for process control in practice because of the difficulty associated with building an accurate nonlinear model (Morari & Lee, 1999). For a more widespread use of nonlinear models for process control, a control method seamlessly integrated with a *nonlinear system identification* strategy needs to emerge. Currently, popular nonlinear model structures studied in the literature include Volterra series models, block-oriented models such as Hammerstein and Wiener models, bilinear models, and Nonlinear AutoRegressive with eXogenous input (NARX) models (Sjöberg et al., 1995).

In a typical situation, various restrictions in plant testing will result in insufficient information about many parts of the dynamic state space leading to significant variance errors in the model (Su & McAvoy, 1993). Control actions and performance predictions calculated from such a model may not be reliable. Since an empirical nonlinear model is valid only in the regions of the state space where identification data were available, one must exercise a caution in using it

☆ This paper was not presented at any IFAC meeting. This paper was recommended for publication in revised form by Associate Editor B.W. Bequette under the direction of Editor F. Allgower. Funded by NSF (CTS-#0301993).
* Corresponding author. Tel.: +1 404 385 2148; fax: +1 404 894 2866.
E-mail addresses: jongmin.lee@chbe.gatech.edu (J.M. Lee), jay.lee@chbe.gatech.edu (J.H. Lee).

0005-1098/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.automatica.2005.02.006

for model-based control. One option is to restrict the control moves to those keeping the system within "trust" regions of the state space (i.e., regions well "covered" by the identification data). The question is how to do this in a systematic and computationally amenable manner. Leonard, Kramer, and Ungar (1992) proposed a radial basis function network (RBFN) that computes the prediction reliability in terms of extrapolation and interpolation. The *validity index* was computed using clustered points with Parzen density estimator (Parzen, 1962) for detecting over-extrapolation. Though developed for the RBFN-type structures, the idea of using a probability density estimator to define "reliable" search space seems attractive in the general case. Nevertheless, accommodation of such a measure of confidence within predictive control should be difficult and computationally demanding, as such measures must be applied to all time steps in the prediction window. Tsai, Chu, Jang, and Shieh (2002) proposed a coordinated architecture between a conventional MPC and an additional neural adaptive controller (NAC), which uses NAC for unexplored regions as detected by a density estimator. This approach can be inefficient and unreliable for highly nonlinear systems in that the structure requires additional training for the NAC design and simply combines control actions of MPC and NAC in a linear manner.

In an attempt to address several shortcomings of the current MPC formulation including its inability to account for model uncertainty properly, we recently suggested an alternative approach based on *approximate dynamic programming* (ADP) (Lee & Lee, 2004). This approach has its roots in the Artificial Intelligence (AI) literature and closely follows the ideas of reinforcement learning (RL) (Sutton & Barto, 1998) and neuro-dynamic programming (NDP) (Bertsekas & Tsitsiklis, 1996). The approach attempts to solve the stochastic optimal control problem through dynamic programming but only approximately and within limited regions of state space. The usual barrier of *curse-of-dimensionality* is alleviated by employing closed-loop simulations and function approximation.

In this paper, building upon our previous work on ADP, we present two different ADP-based strategies for nonlinear control using input–output data alone. In the first strategy, one builds and uses an empirical nonlinear model just like in the conventional MPC approach but in both the cost-to-go approximation and on-line control calculation, extrapolation is controlled by adding a penalty term designed based on the local data distribution. This way, the search for optimal control moves is restricted to keep the system state within the parts of the state space with adequate data densities. The second one aims at improving a given control policy in a continual manner without building a model. For both approaches, a localized approximator is employed for the cost-to-go function approximation. The approaches are illustrated with a CSTR example to demonstrate the difference with the conventional MPC approach.

2. Background materials

2.1. Dynamic programming and cost-to-go function-based control

Dynamic programming (DP) is a general approach for solving sequential multi-stage optimization problems. The objective of DP is to obtain a so-called 'cost-to-go' function. The cost-to-go function is typically a discounted infinite sum of single-stage costs starting from a given state under a certain control policy, which maps the state to the control action. The α-optimal cost-to-go function, J^*, is the cost-to-go function under the optimal control policy (μ^*):

$$J^*(x(0)) = \sum_{k=0}^{\infty} \alpha^k \phi(x(k), \mu^*(x(k))), \tag{1}$$

where $\alpha \in [0, 1)$ is a discount factor and ϕ is a single-stage cost. The optimal cost-to-go function is unique and satisfies the following *Bellman equation* (Bellman, 1957):

$$J^*(x) = \min_u \{\phi(x, u) + \alpha J^*(F_{t_s}(x, u))\}, \tag{2}$$

where $F_{t_s}(x, u)$ denotes the state obtained by propagating the state transition equation from the initial state x for one sample time interval with a constant input of u. Note that the above equation is for a discounted-infinite-horizon case. Finite-horizon formulation is also simple to derive but it will not be covered in this paper. With the optimal cost-to-go function ($J^*(x)$) obtained from the off-line calculation, one can solve on-line the following single-stage optimal control problem, which is equivalent to the infinite-horizon problem:

$$u = \arg\min_u \{\phi(x, u) + \alpha J^*(F_{t_s}(x, u))\}. \tag{3}$$

The benefit of using the cost-to-go function in the optimal control calculation is twofold. First, a multi-stage optimization problem is reduced to a single-stage one. This can not only reduce the on-line computational load, but it can also facilitate the incorporation of some confidence measure of a model into the control action calculation. Second, the cost-to-go can be defined to account for the effect of future uncertainty and feedback in a proper 'closed-loop' manner, whereas conventional MPC assumes open-loop control at future time points in the prediction, which can lead to overly conservative control actions. Formulation of stochastic optimal control as a DP can be done in a manner similar to the above but with the additional use of the expectation operator and an information vector. Despite the generality, DP suffers from the curse-of-dimensionality, which makes the method unsuitable for most practical problems.

2.2. Approximate dynamic programming

Whereas the classical DP approach tries to determine the optimal cost-to-go function for an entire state space through

exhaustive sampling of the state space, the ADP approach attempts to solve the optimality equation only approximately within limited confines of the state space, chosen by running simulations with a number of known suboptimal policies under all plausible operating conditions and building an approximate cost-to-go function using some function approximator. The cost-to-go function approximation is then improved through iteration of the Bellman equation (as in 'value iteration') or the iteration between the Bellman equation and the policy evaluation (as in 'policy iteration'). However, this is done only for those regions of the state space "visited" by the simulated suboptimal policies in the simulation, thereby significantly reducing the computational load.

The obstacle to doing this is that since one does not have the optimal controller, the working state space regime of the optimal controller for a given range of disturbances and set-point changes cannot be identified a priori. Hence, one attempts to construct a superset that includes the relevant regions by closed-loop simulations with judiciously chosen suboptimal policies. If one is not successful, the resulting control policy will be only suboptimal, even though it may still represent a great improvement from the starting suboptimal policies. Since the state space for most chemical process control problems is continuous, some function approximation scheme based on sampled data should be employed for the cost-to-go approximation.

In summary, the proposed scheme is characterized by simulation, function approximation, and iterative improvement of a control policy within limited regions of state space. The converged cost-to-go function should yield an approximately optimal (or at least improved) control policy. The following is the basic synopsis of the suggested approach:

(1) Perform closed-loop simulations (or identification experiments) with a chosen suboptimal control policy (μ^0) under all representative operating conditions. μ^0 can be a manual control strategy, a PI controller, MPC, etc.
(2) Using the simulation (identification) data, calculate the infinite (or finite) horizon cost-to-go (J^{μ^0}) for each state visited during the simulation according to

$$J^{\mu^0}(x(k)) = \sum_{t=0}^{\infty} \alpha^t \phi(x(k+t), u(k+t)). \tag{4}$$

(3) Construct a function approximator for the data to approximate the cost-to-go as a function of continuous state variables, denoted hereafter as $\tilde{J}^{\mu^0}$.
(4) To improve the cost-to-go approximation, perform the value or policy iteration until convergence. In this work, we use the value iteration algorithm. The difference with the policy iteration algorithm can be found in Kaisare, Lee, and Lee (2002). In each iteration loop of the value iteration, we calculate J^{i+1} for each sample point of x by solving

$$J^{i+1}(x) = \min_u \{\phi(x,u) + \alpha \tilde{J}^i(F_{t_s}(x,u))\}, \tag{5}$$

where the superscript i denotes the ith iteration step and $\tilde{J}^0 = \tilde{J}^{\mu^0}$. Once the cost-to-go values are updated for all the states, we then fit another function approximator to the x vs. $J^{i+1}(x)$ data. Note that F_{t_s} would be defined by the identified NARX model in our case.
(5) Once the cost-to-go iteration converges, one can implement the control policy as defined by the following optimization problem:

$$u(k) = \arg\min_{u(k)} [\phi(x(k), u(k)) + \alpha \tilde{J}^*(x(k+1))], \tag{6}$$

where $\tilde{J}^*$ represents the converged cost-to-go and $x(k+1) = F_{t_s}(x(k), u(k))$.

3. Empirical model-based ADP approach: J-learning

One potential problem of using a cost-to-go function approximated from simulation or identification data is that it is only accurate within those regions of the state space with adequate data densities. During on-line optimization, however, the optimizer may push the solution to other regions of inadequate data densities based on a cost-to-go estimate that is not trustworthy. Thus, considering the accuracy of a cost-to-go estimate (based on the local data density) in the control move optimization is essential for the success of the suggested ADP approach.

We found that global approximators such as neural networks can amplify an approximation error in an unpredictable manner during the off-line iteration, whereas a certain class of localized function approximators like the k-nearest-neighbor averager and the kernel-based local averager show nice convergence behavior (Lee, 2004). The local averaging scheme has another advantage that quantifying the confidence in the cost-to-go approximation can be done very naturally on the basis of the local data distribution. The measure of confidence can be used to define a 'risk' term, which can be included in the objective function to discourage the controller from venturing into the regions of state space for which the confidence is low. This way, the optimizer can be coaxed into using the empirical model only in the regions of the state space with adequate data densities, in both the value iteration step and the on-line optimization.

In this paper, we employ a distance-weighted k-nearest-neighbor approximation scheme of

$$\tilde{J}(x) = \sum_{x_j \in N_k(x)} w_j J(x_j), \quad w_j = \frac{1/d_j}{\sum_j 1/d_j}, \tag{7}$$

where d_j is the distance from x to x_j, and $N_k(x)$ is the set composed of the k-nearest neighbors of x. Since the cost-to-go values of the neighbors are weighted proportionally to their distances in calculating the average, the choice of distance metric has a large effect on the cost-to-go estimation. Typically, the Euclidean distance is used in the case of

continuous variables:

$$d_j = \sqrt{(x - x_j)^{\mathrm{T}} W (x - x_j)}, \tag{8}$$

where W is a diagonal feature weighting matrix assigning a weight to each dimension. This feature weighting matrix can be used to emphasize dimensions that are more important than others.

It can be proved that this local averaging scheme guarantees the convergence of the off-line Bellman iteration (Lee, 2004); however, it can still leave significant bias in regions of the state space where the data density is inadequately low. Hence it is important to use the resulting cost-to-go approximation cautiously by considering the data distribution associated with each approximation. To estimate the local data density, a multi-dimensional Parzen probability density estimator (Parzen, 1962) is employed. Suppose that we have a training data set Ω and a new query point x. An estimate of the Parzen density at x, denoted here by $f_\Omega(x)$, is obtained as a sum of kernel functions placed at each sample in Ω:

$$f_\Omega(x) = \frac{1}{N\sigma^{m_0}} \sum_{j=1}^{N} K\left(\frac{x - x_j}{\sigma}\right), \tag{9}$$

where $x, x_j \in \mathbb{R}^{m_0}$, m_0 is the state dimension, $K(\cdot)$ is a selected kernel function, and σ is a user-given bandwidth parameter. Note that N is the number of neighbors used for estimating the cost-to-go. A well-known and widely used kernel is the multivariate Gaussian distribution function:

$$K\left(\frac{x - x_j}{\sigma}\right) = \frac{1}{(2\pi\sigma^2)^{m_0/2}} \exp\left(-\frac{\|x - x_j\|_2^2}{2\sigma^2}\right). \tag{10}$$

To use the cost-to-go approximator cautiously, we incorporate into the cost-to-go a quadratic penalty-term based on $f_\Omega(x)$:

$$\tilde{J}_{\mathrm{new}}(x) \Leftarrow \tilde{J}_{\mathrm{old}}(x) + J_{\mathrm{bias}}(x), \tag{11}$$

where

$$J_{\mathrm{bias}}(x) = AH\left(\frac{1}{f_\Omega(x)} - \rho\right) \cdot \left[\frac{1/f_\Omega(x) - \rho}{\rho}\right]^2, \tag{12}$$

where $H(\cdot)$ is the Heaviside step function, A is a scaling parameter, and ρ is a threshold value. The parameters are determined by the following procedure:

(1) Determine the bandwidth parameter σ in Eq. (9) by considering the distance range of collected data.
(2) The buffer zone, inside which no penalty term is assigned, is determined by setting $\|x - x_j\|_2 = \sigma$ in Eq. (10), calculating the corresponding f_Ω from Eq. (9), and then setting ρ to be the reciprocal of this value.
(3) A, which controls the rate of increasing penalty term, is adjusted to give $J_{\mathrm{bias}}(x) = J_{\max}$ at f_Ω corresponding to $\|x - x_j\|_2 = 3\sigma$ in Eq. (12), where $J_{\max}$ is the largest cost-to-go value observed from the simulation step.

Since a larger σ value allows for more extrapolations, it is preferable to design σ to include a "reasonable" number of data points inside the hypersphere. The other parameters such as A and $J_{\max}$ are not critical to the resulting performance because the controller uses cost-to-go estimates only for selecting a successor state with the smallest cost-to-go value. This discourages the optimizer from driving the system into unexplored regions where the data densities are very low. The procedure is described as follows:

(1) Perform closed-loop identification experiments in all possible operating regions by injecting dither signals into the control actions.
(2) Identify a NARX model by fitting a parameterized structure (e.g., neural network, polynomial, etc.) to the data.
(3) Perform the value iteration with the identified model and the initial cost-to-go data until the cost-to-go values for all the visited states converge. To bound the cost-to-go in the off-line iteration steps, the additive penalty term is set as $J_{\max}$ whenever $\tilde{J}(x) \geqslant J_{\max}$.
(4) In on-line control, the converged $\tilde{J}(x)$ is used to calculate the control action according to Eq. (6).

Since the cost-to-go function represented by the local averager is non-smooth in general, manipulated variable u is discretized into a set of values for the global optimization in order to avoid local minima in solving Eqs. (5) and (6).

4. Model-free approach: Q-learning

In this section, we propose a different approach to designing a cost-to-go function based controller, which does not involve identifying an explicit input–output model. The approach allows for a continual improvement of a given control policy through periodic updates based on obtained on-line data. In order to learn a control policy in the absence of an input–output model, the definition of cost-to-go function needs to be changed slightly. Whereas the cost-to-go for the previous approach has the state vector as a sole argument, the cost-to-go for the new approach, which we refer to as the Q-function, maps a state and action pair to the cost-to-go value. The optimal Q-function satisfies

$$Q^*(x(k), u(k)) = \phi(x(k), u(k)) + \alpha \min_{u' \in \mathscr{U}} Q^*(x(k+1), u'), \tag{13}$$

where $\mathscr{U}$ is a set of all possible actions. Eq. (13) is equivalent to

$$Q^*(x(k), u(k)) = \phi(x(k), u(k)) + \alpha J^*(x(k+1)). \tag{14}$$

Eqs. (13) and (14) imply that Q value is the sum of all the single-stage costs over the infinite horizon starting with a given state and action pair assuming the optimal control policy is enforced from the next time step on.

This approach was originally proposed within the AI community (Watkins & Dayan, 1992) to solve discrete Markov

decision processes (MDPs) where the number of state and action pairs is finite. The conventional way to solve for the optimal Q function is to iterate Eq. (13) by performing a large number of on-line experiments to allow for multiple visits to a same state with different action values. General update rule of the Q-value is

$$Q^{i+1}(x(k),u(k)) = (1-\gamma)Q^i(x(k),u(k)) + \gamma\left\{\phi(x(k),u(k)) + \alpha \min_{u'\in\mathscr{U}} Q^i(x(k+1),u')\right\}, \quad (15)$$

where γ is a learning rate parameter between 0 and 1 (Watkins & Dayan, 1992).

However, the conventional Q-learning is not well suited for process control problems due to the continuous nature of typical state and action spaces. States collected from transient trajectories will tend to be distinct and multiple visits to a same state unlikely even with a very large number of experiments. In other words, different control policies and randomization do not ensure multiple visits to exact same states.

Recently, several modified Q-learning methods that are capable of handling continuous state and action spaces have been proposed. Most of them are based on discretizing state and action spaces and then posing the problem as a discrete MDP. However, the discretization approach is limited to small-size problems and the result can be very sensitive to the discretization method (Takeda, Nakamura, Imai, Ogasawara, & Asada, 2000). One noteworthy work for continuous space Q-learning was proposed by Smart and Kaelbling (2000), who named it the HEDGER algorithm. The main idea is to use existing approximated Q-values for training *neighboring* Q-values and to use a hyper-elliptic hull to prevent extrapolation. The strategy of updating neighboring points based on distance metric is attractive, but the algorithm has some disadvantages when applied to process control problems. Not only is the task of building the independent variable hull (IVH) at each decision point computationally very expensive but the conservativeness of IVH can oftentimes result in no control action taken by the controller.

Based on this consideration, we propose a modified Q-learning algorithm better suited for process control problems. Important features of the modified approach are:

- A local averager is employed for the approximation of the Q values.
- In the implementation of an approximated Q function, the same quadratic penalty adjustment as in Eq. (11) (but with J replaced by Q) is made. The distance metric is based on the concatenated vector composed of the state and action vectors.
- To update the Q values in the memory, the exact Q values are calculated on-line.

The resulting Q-learning procedure for learning an improved control policy in the absence of a model can be summarized as follows:

(1) Perform closed-loop experiments in all possible operating regimes by injecting dither signals into the control actions.
(2) Estimate initial Q values for each visited state-action pair according to

$$Q^0(x(k),u(k)) = \sum_{t=k}^{\infty} \alpha^{t-k}\phi(x(t),u(t)). \quad (16)$$

Based on these values, design a local averager (e.g., distance-weighted k-nearest-neighbor averager), of which the input argument is a concatenated vector of state-action pair. A feature weight can be assigned to each component of the vector to make the prediction more accurate.

$$\tilde{Q}(z) = \sum_{z_j\in N_k(z)} w_j Q(z_j), \quad (17)$$

where $z=[x\ \ u]^{\mathrm{T}}$, $w_j=(1/d_j)/\sum_j 1/d_j$ and d_j is given by Eq. (8).
(3) Perform a closed-loop experiment under the following policy based on the current approximation $\tilde{Q}^i$:

$$u(k) = \arg\min_{u\in\mathscr{U}} \tilde{Q}^i(x(k),u) \quad (18)$$

with the penalty term added to $\tilde{Q}^i$ as before.
(4) Calculate the Q value (Q^{on}) for each visited state and action pair (x_0,u_0) from the on-line experiment according to

$$Q^{\mathrm{on}}(x_0,u_0) = \sum_{t=0}^{\infty} \alpha^t \phi. \quad (19)$$

(5) Update the Q values for (x,u) in the storage using the new information Q^{on}

$$Q^{i+1}(x,u) = Q^i(x,u) + \gamma\{Q^{i,\mathrm{on}}(x_0,u_0) - Q^i(x,u)\}, \quad (20)$$

where i is the off-line update index, and the learning rate parameter γ is assigned as follows:

- If (x_0,u_0) is the same point (within some tolerance) with (x,u)

$$\gamma = \frac{1}{\sqrt{i+1}}. \quad (21)$$

- Otherwise, add the point (x_0,u_0) to the memory and update the Q-values of neighboring points, $(x,u)\in N_{k_u}(x_0,u_0)$, using the distance-weighted factor:

$$\gamma = \frac{1/d_j}{\sum_{j=1}^{k_u} 1/d_j}, \quad (22)$$

where $N_{k_u}(x_0, u_0)$ is the set composed of the k_u-nearest neighbors of (x_0, u_0).

(6) Repeat the procedure from step (3) until no appreciable improvement is observed.

5. Simulation example: CSTR

In this section, an example of CSTR with a first-order exothermic reaction adopted from Hernández and Arkun (1993) is considered. The proportional-integral (PI) controllers, the successive linearization based MPC (slMPC) method described in (Lee & Ricker, 1994), nonlinear programming-based MPC (NMPC), J-learning-based controller, and Q-learning-based controller are compared to illustrate some key aspects of the suggested approaches.

The dynamic equations of the system are given in dimensionless form as

$$\begin{aligned}\dot{x}_1 &= -x_1 + D_a(1-x_1)\exp\left(\frac{x_2}{1+x_2/\varphi}\right),\\ \dot{x}_2 &= -x_2 + BD_a(1-x_1)\exp\left(\frac{x_2}{1+x_2/\varphi}\right) + \beta(u-x_2),\\ y &= x_1,\end{aligned} \tag{23}$$

where x_1 and x_2 are the dimensionless reactant concentration and reactor temperature, respectively. The input u is the cooling jacket temperature. D_a, φ, B, and β are Damköhler number, dimensionless activation energy, heat of reaction, and heat-transfer coefficients, respectively. With the choice of the following parameters (D_a=0.072, φ=20.0, B=8.0, β=0.3), the system shows three steady states, the middle one of which is unstable. The control objective is to take the system from a stable equilibrium point (x_1=0.144, x_2=0.886, u=0.0) to an unstable one ($x_1 = 0.445$, $x_2 = 2.7404$, $u = 0.0$).

5.1. Identification

A NARX structure with a feed-forward neural network was identified using the input–output data obtained from closed-loop simulations with a PI controller. With the dimensionless sample time of 0.5, the output of next time step $y(k+1)$ was found to be a function of $[y(k), \ldots, y(k-3), u(k), \ldots, u(k-3)]$ (Hernández & Arkun, 1993). A state space realization of the NARX model and the design of a model predictive controller with extended Kalman filter can also be found in Hernández and Arkun (1993). For the controller design, the state vector was constructed as $x(k) = [y(k), \ldots, y(k-3), u(k-1), \ldots, u(k-3)]$.

To cover the pertinent operating ranges, different controller gains were used with the set-point of 0.4450. The selected gain values were 9, 6.75, and 4.5 with the same integral time of 83.3 (sample time). For each closed-loop experiment, the input was dithered at each sample time with a random noise generated from the uniform distribution within [−0.03 0.03]. A set of 12 simulations was conducted by changing the set-point from the low steady state to the unstable one. From the simulations, we collected 2820 input–output data points. The identification data (x_1 vs. x_5) are plotted in Fig. 2.

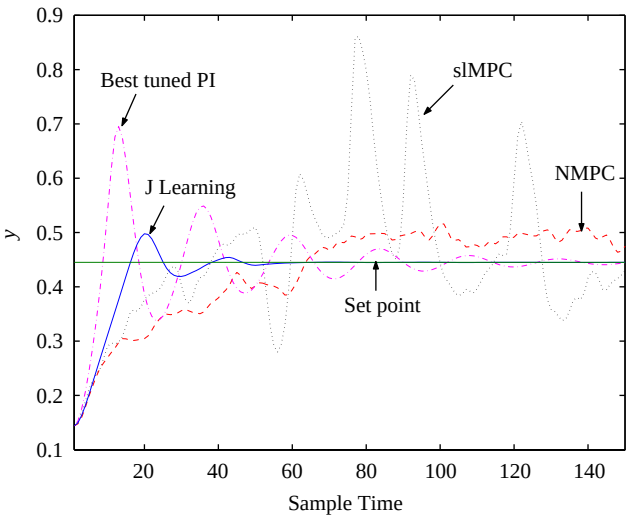

Fig. 1. Comparison of regulation performances.

The neural network we fitted has seven hidden nodes with eight inputs, $x(k)$, $u(k)$, and one output, $y(k+1)$. The parameters were identified using the MATLAB Neural Network Toolbox (Demuth & Beale, 2002) with the fitting tolerance (MSE) set as 1e-5.

5.2. Model predictive control

We tested both the slMPC and NMPC coupled with an EKF estimator. With a prediction horizon of 7 and a control horizon of 1, the MPCs solve the following objective function on-line at each sample time k:

$$\min_{u(k)}\left[\sum_{t=1}^{7} 50\{0.4450 - y(k+t)\}^2 + \Delta u(k)^2\right]. \tag{24}$$

The regulation performances are shown in Fig. 1. We can see that the system could not be regulated by either of the controllers. Larger control horizon choices were found to yield worse performances, probably due to the optimizer settling on local minima. Fig. 2 shows that the poor regulation performance of the MPC is due to the extrapolation of the identified model to unexplored regions of the state space. It is noteworthy that *the output and input weights had to be detuned significantly in order to achieve closed-loop stability*. The ratio of the output weight to the input weight had to be decreased to 5.

5.3. J-learning approach

For the 2820 data, the value iteration was performed with the k-nearest-neighbor approximator, which averages four neighboring points (k=4) for the cost-to-go approximation.

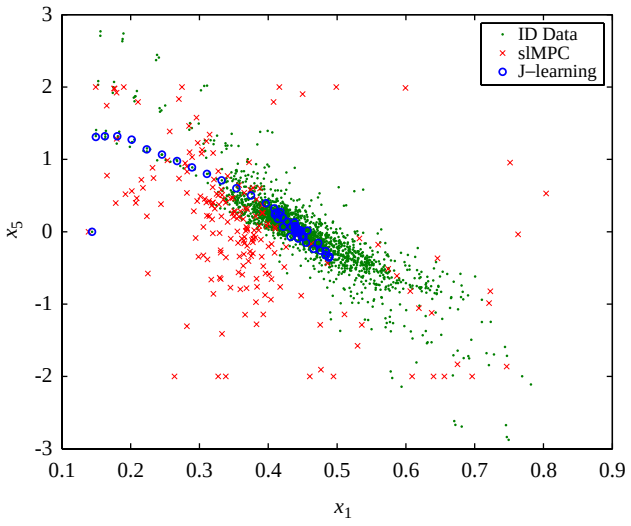

Fig. 2. Comparison of on-line state trajectories: plot of x_1 vs. x_5.

Using a discount factor of 0.98, the initial cost-to-go values were calculated by Eq. (4) with the following definition of one-stage cost:

$$\phi(x(k), u(k)) = 50(0.4450 - y(k+1))^2 + \Delta u(k)^2. \quad (25)$$

Hence, with the ADP-based method, we are attempting to derive a much less detuned controller that still maintains closed-loop stability. The convergence criterion used for the value iteration was

$$e_{\text{rel}} = \left\| \frac{J^i(x_k) - J^{i-1}(x_k)}{J^{i-1}(x_k)} \right\|_\infty < 0.01, \quad (26)$$

where $k = 1, \ldots, 2820$ and i is the iteration index. The off-line value iteration converged after 31 steps, during which e_{rel} decreased monotonically. The parameters of the penalty function were set as $\sigma = 0.1587$ (1% of normalized distance range), $\rho = 6.6 \times 10^{-9}$, $A = 0.0696$, and $J_{\max} = 200$. Fig. 1 and Table 2 show the improvement in performance from the starting control policies (PI controllers), and Fig. 2 illustrates that the suggested strategy uses the model in the vicinity of the data and avoids unreasonable extrapolations.

5.4. Q-learning approach

Using the same starting PI controllers and input dither signals, 3256 points of state and action pairs were collected. Initial Q values for each state and action pair were estimated by Eq. (16) with α of 0.98. For the Q function approximation, the k-nearest neighbor approximator with $k = 4$ was employed and $W = \text{diag}[5\ 3\ 3\ 2\ 3\ 2\ 2\ 5]$ was used to assign more weights to the current input and output values. $k_u = 15$ was used for action sampling and update of neighboring Q values based on the additional on-line information. The parameters of the penalty function were set as $\sigma = 0.1697$, $\rho = 1.2 \times 10^{-9}$, $A = 0.0696$, and $Q_{\max} = 200$.

Table 1
Number of data points in the memory and on-line performances (undiscounted ∞-horizon cost) at some iteration steps of the Q-learning

Iteration	Data	$\sum_{k=0}^{\infty} \phi$	Iteration	Data	$\sum_{k=0}^{\infty} \phi$
1	3256	44.76	10	4809	29.72
5	3990	34.09	20	6418	27.53

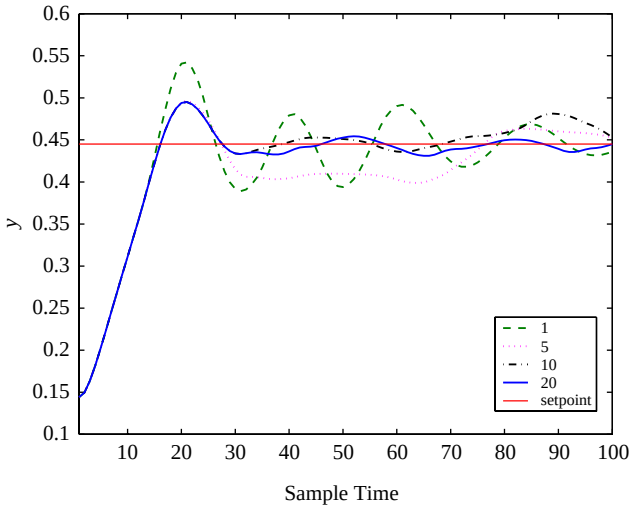

Fig. 3. Evolutionary improvement of regulation performance under Q-learning.

Table 2
Comparison of the closed-loop performance of control polices: undiscounted ∞-horizon cost

Controller	$\sum_{k=0}^{\infty} \phi$	Controller	$\sum_{k=0}^{\infty} \phi$
PI (Kc = 4.5)	60.9	slMPC	∞
PI (Kc = 6.75)	43.8	NMPC	∞
PI (Kc = 9.0)	58.0	slMPC (detuned)	165
J-learning	27.2	NMPC (detuned)	42.0
Q-learning	27.5		

As mentioned earlier, this interactive learning between on-line experiment and off-line redesign of the Q function (and therefore the controller) in the absence of a model involves the update of new information at each iteration, which implies a gradual increase in the number of data points stored in the memory. Table 1 indicates about 100% increase in the number of data points after 20 iteration steps, which is quite reasonable.

Fig. 3 shows oscillatory behavior in the early phase of learning, but after a while the performance improvement is gradual and reaches very good performance after 20 iteration steps. The resulting state trajectory also interpolated well to the target point. Table 2 shows that the ADP schemes improved the starting control policies (PI controllers) while avoiding the extrapolation problem seen in the control by the MPCs.

References

Bellman, R. E. (1957). *Dynamic programming*. Princeton, New Jersey: Princeton University Press.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming*. Belmont, MA: Athena Scientific.

Demuth, H., & Beale, M. (2002). *Neural network toolbox user's guide (MATLAB)*. Natick, MA: The MathWorks Inc.

Hernández, E., & Arkun, Y. (1993). Control of nonlinear systems using polynomial ARMA models. *AIChE Journal*, *39*(3), 446–460.

Kaisare, N. S., Lee, J. M., & Lee, J. H. (2002). Comparison of policy iteration, value iteration and temporal difference learning. In *AIChE Annual Meeting*, Indianapolis, IN.

Lee, J. M. (2004). *A study on architecture, algorithms, and applications of approximate dynamic programming-based approach to optimal control*. Ph.D. thesis, Georgia Institute of Technology.

Lee, J. M., & Lee, J. H. (2004). Simulation-based learning of cost-to-go for control of nonlinear processes. *Korean Journal of Chemical Engineering*, *21*(2), 338–344.

Lee, J. H., & Ricker, N. L. (1994). Extended Kalman filter based nonlinear model predictive control. *Industrial and Engineering Chemistry Research*, *33*, 1530–1541.

Leonard, J. A., Kramer, M. A., & Ungar, L. H. (1992). A neural network architecture that computes its own reliability. *Computers and Chemical Engineering*, *16*, 819–835.

Morari, M., & Lee, J. H. (1999). Model predictive control: Past, present and future. *Computers and Chemical Engineering*, *23*, 667–682.

Parzen, E. (1962). On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, *33*, 1065–1076.

Sjöberg, J., Zhang, Q., Ljung, L., Benveniste, A., Delyon, B., Glorennec, P.-Y., Hjalmarsson, H., & Juditsky, A. (1995). Nonlinear black-box modeling in system identification: A unified overview. *Automatica*, *31*(12), 1691–1724.

Smart, W. D., & Kaelbling, L. P. (2000). Practical reinforcement learning in continuous spaces. In *Proceedings of the 17th international conference on machine learning* (pp. 903–910). San Francisco, CA: Morgan Kaufmann.

Su, H. T., & McAvoy, T. J. (1993). Integration of multilayer perceptron networks and linear dynamic models: A Hammerstein modelling approach. *Industrial and Engineering Chemistry Research*, *32*, 1927–1936.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Takeda, M., Nakamura, T., Imai, M., Ogasawara, T., & Asada, M. (2000). Enhanced continuous valued Q-learning for real autonomous robots. *Advanced Robotics*, *14*(5), 439–442.

Tsai, P.-F., Chu, J.-Z., Jang, S.-S., & Shieh, S.-S. (2002). Developing a robust model predictive control architecture through regional knowledge analysis of artificial neural networks. *Journal of Process Control*, *13*, 423–435.

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*, 279–292.

Jong Min Lee received his B.S. degree in Chemical Engineering from Seoul National University, Seoul, Korea in 1996, and his Ph.D. degree in Chemical and Biomolecular Engineering from Georgia Institute of Technology, Atlanta, in 2004. He is currently a postdoctoral researcher in the School of Chemical and Biomolecular Engineering at Georgia Institute of Technology, Atlanta. His current research interests are in the areas of optimal control, dynamic programming and reinforcement learning.

Jay H. Lee obtained his B.S. degree in Chemical Engineering from the University of Washington, Seattle, in 1986, and his Ph.D. degree in Chemical Engineering from the California Institute of Technology, Pasadena, in 1991. From 1991 to 1998, he was with the Department of Chemical Engineering at the Auburn University, AL, as an Assistant Professor and an Associate Professor. From 1998 to 2000, he was with the School of Chemical Engineering at Purdue University, West Lafayette, as an Associate Professor. Currently, he is a Professor in the School of Chemical and Biomolecular Engineering and a director of the Center for Process Systems Engineering at the Georgia Institute of Technology, Atlanta. He has held visiting appointments at E. I. Du Pont de Numours, Wilmington, in 1994 and at Seoul National University, Seoul, Korea, in 1997. He was a recipient of the National Science Foundation's Young Investigator Award in 1993. His research interests are in the areas of system identification, robust control, model predictive control and nonlinear estimation.