Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

Parametric Approximation of Piecewise Quadratic Value Functions for the Control of Complex Systems

Hussam Nosair * **Jong Min Lee** *,[1]

* *Chemical and Materials Engineering, University of Alberta, Edmonton, AB, Canada T6G 2G6.*

Abstract: We present a new technique for approximate dynamic programming that is suitable for control of large-scale systems with complex dynamics. The approach improves closed-loop performances from a starting control policy by incrementally updating a value function on-line based on the Bellman's optimality equation. The value function is approximated as a map between the state and the associated cost-to-go value to circumvent the "curse-of-dimensionality." The approximation method uses piecewise quadratic representations of the value function, and can considerably reduce the computational requirement compared to the instance-based methods, which store all the historical data and retrieve them for calculating optimal control actions. Hybrid and nonlinear examples are included to demonstrate the applicability of the approach.

1. INTRODUCTION

With complex systems increasingly being introduced in the process industries, system complexity is currently a major challenge for process control. Among them are nonlinear systems, multi-scale systems, and hybrid systems that include both continuous and discrete states.

Model predictive control (MPC) has become the most popular control technique for multivariable systems with constraints since its arrival in the 1970s. It uses a dynamic model to predict the future output behavior and compute a sequence of control moves that minimize the output deviation from a set-point trajectory. Feedback is incorporated by observing the system's response, and the optimization is successively repeated at each sample time.

Despite its growing number of novel applications in nontraditional areas such as aeronautics, the automotive industry and medicine [Morris, 2003], there are several major concerns in applying MPC to complex systems. First, it is difficult to build an accurate dynamic model. Second, MPC requires the solution of a nonlinear or a mixed-integer program where the number of optimization variables, i.e., control moves, increases with the size of control horizon. Third, the open-loop optimization scheme of MPC cannot take into account the future interplay between uncertainty and estimation in the optimal control calculation. Modified MPCs with different worst case formulations still lead to overly conservative feedback control law for uncertain systems [Lee and Yu, 1997].

To address the inherent MPC's limitations, Lee and coworkers [Lee and Lee, 2005, Lee et al., 2006] proposed an approximate dynamic programming (ADP)-based strategy suited for process control problems. It takes advantage of the dynamic programming (DP)'s closed-loop optimization and potential reduction of on-line computational burden while circumventing Bellman's "curse of dimensionality," the exponential increase in computational and storage requirements associated with addition of extra dimensions to the state space.

The proposed ADP strategy solves the Bellman's optimality equation *off-line* for the state points sampled from closed-loop simulations or experiments to estimate the optimal value function, a map between a state and minimum achievable total cost starting from the state. Since typical control problems involve a continuous state space, a function approximator is used to estimate the "cost-to-go" value for a given state. The ADP approach uses an instance-based learning scheme, e.g., k-nearest neighbour averaging, because it guarantees monotonic convergence of the off-line iteration step [Gordon, 1995, Lee et al., 2006].

The instance-based approximator stores *all* the state-value data in the memory and retrieves them to evaluate the cost-to-go value for a state point when necessary. Two critical issues in using such an approximator are the prohibitive storage requirement and the computational burden in searching the neighbours and averaging the associated values. In addition, the local averaging scheme makes the optimization non-convex. Parametric approximation can be an alternative approach, because it does not store and retrieve the data in the optimization. Fitting a global nonlinear function like a neural network, however, has been shown to suffer from the instability of learning and non-convexity of optimization [Gordon, 1995, Lee et al., 2006].

In this paper we present a novel approach for approximating the state and cost-to-go value map and utilizing the approximate value function in a more computationally amenable manner. We approximate the value function as piecewise quadratic functions of the state. The rationale is that complex dynamics can be well approximated as a set of piecewise affine functions. With a linear dynamics and

[1] Corresponding author. (Tel: +1-780-492-8092; e-mail: jongmin.lee @ualberta.ca)

a quadratic stage-wise cost, the value function becomes a quadratic function of the state. The optimization in this formulation is convex, and only a small number of parameters are required to encode the cost-to-go as a function of the state.

In the proposed approach, we start with collecting closed-loop data using existing suboptimal controllers. The data points are clustered into subsets to each of which a quadratic value function is fitted. The parameters of each approximate value function is then "incrementally" updated at each sample time while the control actions calculated from the approximate value function is implemented. This can bring a significant improvement of control performance even in the absence of a process model.

2. BACKGROUND

2.1 Dynamic programming and value function-based control

Consider a discrete-time dynamic system in the form of

$$x(k+1) = f(x(k),\ u(k),\ e(k)) \tag{1}$$

where $x(k)$ is a state vector, $u(k)$ is a manipulated input vector, $e(k)$ is some random disturbance at the k^{th} sample time, and f is a state transition function. We define the cost function for an infinite horizon problem given an initial state $x(0)$ and control policy μ as follows:

$$J^{\mu}(x) \triangleq E_{\mu}\left[\sum_{k=0}^{\infty} \alpha^k \phi(x(k), u(k)) \,|x(0) = x\right] \quad \forall x \in \mathcal{X} \tag{2}$$

where $\alpha \in (0,\ 1)$ is a discount factor, ϕ is a single-stage cost function, $\mathcal{X}$ is the set of all possible states, and E_{μ} is the conditional expectation taken under the control policy μ, which maps a state to a control action. The α-optimal "value function" J^* is then defined as

$$J^*(x) \triangleq \min_{\mu = \{u(0),\ u(1),\ \cdots\}} J^{\mu}(x) \quad \forall x \in \mathcal{X} \tag{3}$$

, which represents the minimum expected discounted cost when starting from a state x and always following an optimal control policy.

The optimal value function is unique and satisfies the following Bellman equation [Bellman, 1957].

$$J^*(x) = \min_{u \in \mathcal{U}} E\left[\phi(x,u) + \alpha J^*(f(x,u,e))\right] \quad \forall x \in \mathcal{X} \tag{4}$$

where the set of all possible actions is represented by $\mathcal{U}$. With the optimal value 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:

$$\mu^*(x) = u^* = \arg\min_{u \in \mathcal{U}} E\left[\phi(x,u) + \alpha J^*(f(x,u,e))\right] \tag{5}$$

Except for a few special cases (e.g., linear quadratic optimal control), a closed-form solution of the Bellman equation is not available. Hence, numerical approaches like "value iteration" or "policy iteration" algorithms should be used with exhaustive sampling or discretization of the state space [Bertsekas, 2000]. Unfortunately, the computational burden of both conventional approaches quickly becomes prohibitive as the dimensions of state and control action increase.

2.2 Model-free learning of the value function

The value function-based controller can be designed even in the absence of a process model, f, by mapping a state and control action pair to the value function, referred to as the Q-function [Watkins and Dayan, 1992]. The optimal Q function is defined as

$$\begin{aligned} Q^*(x(k),u(k)) \triangleq \min_{u(k+1),\ u(k+2),\cdots} & E\left[\phi(x(k),u(k))\right. \\ & \left. + \alpha\phi(x(k+1),u(k+1)) + \cdots\right] \end{aligned} \tag{6}$$

, and it satisfies

$$\begin{aligned} Q^*(x(k),u(k)) = & \ \phi(x(k),u(k)) \\ & + \alpha \min_{u(k+1)\in\mathcal{U}} Q^*(x(k+1),u(k+1)) \end{aligned} \tag{7}$$

where $\mathcal{U}$ is a set of all possible control actions. Eq. (7) is equivalent to

$$Q^*(x(k),u(k)) = \phi(x(k),u(k)) + \alpha J^*(x(k+1)) \tag{8}$$

Eqs. (7) and (8) imply that the optimal 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 can be particularly useful when the identification of a given process is difficult.

A conventional way of solving for the optimal Q function is to iterate on (7) by performing a large number of on-line experiments to allow for multiple visits to a same state with different control actions [Watkins and Dayan, 1992]. However, this is not well suited for process control problems due to the continuous nature of state and control action spaces. States collected from transient trajectories will tend to be distinct and multiple visits to the same state are unlikely even with a large number of experiments.

3. PIECEWISE QUADRATIC APPROXIMATION OF VALUE FUNCTIONS

A proper approximation scheme is needed for handling the computational issues arising from either large or continuous state and action spaces in solving an optimal control problem. It was shown that constructing a global approximate value function using a nonlinear mapping such as neural networks can amplify approximation errors in an unpredictable manner, whereas averaging local values increases the computational burden and storage requirement in learning and using the value function [Lee et al., 2006, Lee and Lee, 2005]. The approach proposed in this work is to construct an approximate value function as piecewise quadratic functions. The advantage of the proposed technique is the reduction of computational complexity and storage requirement due to its local convexity and parametric representation.

3.1 Proposed approach

We start with closed-loop simulations (or operations) with available suboptimal control policies, e.g., a manual control strategy, a PID controller, MPC, etc., under all representative operating conditions. After this,

(1) calculate the initial values for the collected state points $x(k)$ using

$$J^0(x(k)) = \sum_{t=0}^{\infty} \alpha^t \phi(x(k+t), u(k+t)) \tag{9}$$

(2) cluster the data points $(x,\ J^0(x))$ (or $([x,u],\ Q^0(x,u))$ in the model-free learning) to find the valid regions for each quadratic value function.
(3) fit a local quadratic value function to each clustered data set.
(4) select control actions according to the current value function estimate.
(5) update the value function according to the experience at each sample time.
(6) the steps (4) and (5) are iterated.

3.2 Clustering the value data and fitting to quadratic functions

With the usual quadratic term for the stage-wise cost ϕ, a cost-to-go map for a complex system can be approximated locally as quadratic. We first classify the data generated from initial control policies to assign each data point to the local quadratic value function that more likely generated it. Thus, small subsets of $x(k)$ that are close to each other are likely to belong to the same region. Conventional classification procedures find the optimal location of each centroid by minimizing the sum of distances from the centroid to the data points belonging to the cluster. Performing a classification directly in the state space (or state-action space), however, may lead to irrelevant clustering results for approximating the value function; it yields most of the centroids populated in the regions where data density is high, e.g., the region near an equilibrium point in a regulation problem [Hastie et al., 2001]. Ferrari-Trecate et al. [2003] suggested a method that classifies the data in the parameter space for identifying piece-wise affine submodels. However, this method may yield very similar parameter vectors for different local data sets in different regions. This results in indistinctive clusters at classification, which causes difficulties in learning value functions.

To address the difficulties, we classify the data according to a feature vector that concatenates state (or state and action for model-free learning) and parameter vectors. The procedure is as follows:

(1) Build local data sets (LDs) from the original data: For each data point $(x(k),\ J^0(x(k)))$ (or $([x(k),\ u(k)],\ Q^0(x(k),u(k)))$ in the model-free case), we build a local data set $\mathcal{C}_k$ collecting $(x(k),\ J^0(x(k)))$ and the $c-1$ distinct data points $(\bar{x},\ \bar{J})$ that satisfy
$$\|x(k)-\bar{x}\|^2 \leq \|x(k)-\hat{x}\|^2,\ \forall(\hat{x},\ \hat{J}^0) \in \Psi \setminus C_k \quad (10)$$
where $\|\cdot\|$ is the Euclidean norm, $k=1,\ \cdots,\ N$ with N the number of data points in the set Ψ. c is a user-given parameter.
(2) Identify a parameter vector based on each LD by fitting to a quadratic function in the form of
$$\tilde{J}_k(x) = x^T \Pi x + b \quad (11)$$
or
$$\tilde{Q}_k(x,u) = \begin{bmatrix} x^T & u^T \end{bmatrix} \Theta \begin{bmatrix} x \\ u \end{bmatrix} + b \quad (12)$$
where b is a bias term and $k=1,\ \cdots, N$.
(3) Classify the feature vectors composed of states (or state and action pairs) and parameters, and assign a cluster to each original data based on a bijective map between the data points and LDs. The optimal number of clusters, s, is determined by looking at the separation of the regions as well as the fitting performance, $\sum_{k=1}^{N}\left(J(x(k)) - \tilde{J}_i(x(k))\right)^2,\ i \in 1,\cdots,s$. We use support vector machine (SVM) Vapnik [1998] for classification in this work.

3.3 Calculating control actions

Once we have an initial estimate of the value function from existing control policies, we can use the approximate value function to select a greedy control action at each time t by
$$u(t) = \arg\min_{u(t)} E\left[\phi(x(t),u(t)) + \alpha \tilde{J}^0(x(t+1))\right] \quad (13)$$
or
$$u(t) = \arg\min_{u(t)} \tilde{Q}^0(x(t),u(t)) \quad (14)$$
where $\tilde{J}^0 = \left\{\tilde{J}_1^0,\ \cdots,\ \tilde{J}_s^0\right\}$ and $\tilde{Q}^0$ is defined in the same manner.

3.4 Updating the value function

Since the initial cost-to-go values are based on a suboptimal control policy or the combination of starting policies, one can still improve the value function through iterating on the Bellman's equation. In this work, we incrementally update the value function on-line based on "temporal-difference (TD)" [Sutton and Barto, 1998]. The advantages are that a value function can be improved at each time step in an on-line fashion rather than waiting until a simulation or operation episode ends as in classical DP approaches [Bertsekas, 2000] or in the previous ADP approaches [Lee and Lee, 2005]. In addition, the incremental update rule does not require a process model.

We define the one-step TD error after experiencing the stage-wise cost at time t as
$$\delta(t) \triangleq (\phi(x(t),u(t)) + \alpha \tilde{J}_t(x(t+1))) - \tilde{J}_t(x(t)) \quad (15)$$
The function approximators are defined as
$$\tilde{J}(x) = \theta^T \cdot \Phi \quad (16)$$
where θ is the parameter vector for a quadratic value function and Φ is a regressor vector composed of quadratic and linear terms of x and u.

Then the one-step TD update rule for the parameter vector is given by
$$\theta(t+1) = \theta(t) + \gamma\delta(t)\nabla_{\theta(t)}\tilde{J}_t(x(t)) \quad (17)$$
where γ is learning rate and t is the time index in a single episode (or a sample trajectory). Note that the learning rate γ is positive and $\delta(t)\nabla_{\theta(t)}\tilde{J}_t(x(t))$ is the vector with a descent direction of $\tilde{J}$.

Whereas the one-step TD update rule, referred to as TD(0) update, is based on the current difference $\delta(t)$ only, a more general version called TD(λ) updates the cost-to-go values by including the temporal differences of the later states visited in a trajectory with exponentially decaying weights [Sutton and Barto, 1998]. A corresponding on-line update rule is given by
$$\theta(t+1) = \theta(t) + \gamma\delta(t)\nabla_{\theta(t)}\tilde{J}_t(x(t))e_t(\Phi(x(t))) \quad (18)$$

where $e_t(\Phi(x_t))$ is referred to as "eligibility". All eligibilities start out with zeros and are updated at each time t as follows:

$$e_t(\Phi(x)) = \begin{cases} \alpha\lambda e_{t-1}(\Phi(x)) & \text{if } \Phi(x) \neq \Phi(x_t) \\ \alpha\lambda e_{t-1}(\Phi(x)) + \Phi(x) & \text{if } \Phi(x) = \Phi(x_t) \end{cases} \tag{19}$$

Note that the temporal difference term, $\delta(t)$ of (18) appears only if the feature vector $\Phi(x(t))$ has already been observed in a previous time of the episode. The eligibility thereby puts more emphasis on the temporal difference term in recent past.

The update rule for the model-free case can also be derived in a similar manner, and it will be briefly described with the illustrative examples.

4. NUMERICAL EXAMPLES

4.1 Hybrid System

We consider a hybrid system with two modes described by linear discrete-time state space systems. The control objective is to regulate the system at the equilibrium point $(x = 0,\ u = 0)$. In this example, we discuss the model-free learning scheme since it is a more challenging problem for controller design.

Mode 1:

$$\begin{aligned} x(k+1) &= \begin{bmatrix} 0.5207 & 0.7432 \\ 0.8088 & -0.1453 \end{bmatrix} x(k) + \begin{bmatrix} 0.4298 \\ 0.4013 \end{bmatrix} u(k) \\ y(k) &= [\,0\ \ 1\,]\, x(k) \\ &\text{if} \quad x \in \mathcal{X}_1 \end{aligned}$$

Mode 2:

$$\begin{aligned} x(k+1) &= \begin{bmatrix} 0.4734 & 0.6756 \\ 0.7353 & -0.1321 \end{bmatrix} x(k) + \begin{bmatrix} 0.4776 \\ 0.4459 \end{bmatrix} u(k) \\ y(k) &= [\,0\ \ 1\,]\, x(k) \\ &\text{if} \quad x \notin \mathcal{X}_1 \end{aligned}$$

where $\mathcal{X}_1 = \{(x_1, x_2) | -1.5 \leq x_1 \leq 1.5 \text{ or } -1.5 \leq x_2 \leq 1.5\}$.

Closed-loop simulations with PI controllers First, closed-loop simulations were performed with suboptimal control policies consisting of four discrete-time PI controllers. The controller parameters were computed from the step responses of the hybrid system, and they were $(K_c, \tau_I) \in (\{1,\ 10\}, \{1,\ 15\}, \{1.5,\ 10\}, \{1.5,\ 15\}, \{2,\ 10\})$ where K_c is the controller gain and τ_I is the integral time. Initial state points were sampled uniformly from the batch $B = \{(x_{1i}, x_{2i}) | (x_{1i}, x_{2i}) \in \{0.5, 1.5, 2.5, 3.5, 4.5\}\}$, and a PI controller was also selected uniformly among the five controllers. Using a discount factor (α) of 0.99, initial cost-to-go values were calculated by

$$Q^{PID}(x(k), u(k)) = \sum_{t=0}^{\infty} \alpha^t \phi(x(k+t), u(k+t)) \tag{20}$$

where

$$\phi(x(k), u(k)) = y^2(k+1) + u^2(k) \tag{21}$$

We collected 413 data points from the simulations.

Table 1. Learned parameters (θ) for $\tilde{Q}$ after 50 batches (1250 episodes)

Cluster	x_1^2	x_2^2	u^2	x_1x_2	x_1u
1	3.0	22	12.0	1.4	7.1
2	3.8	9.9	10.9	10.4	12.5
3	5.6	6.2	5.2	3.1	7.2
Cluster	x_2u	x_1	x_2	u	1
1	22.1	1.9	2.3	1.8	1.7
2	16.4	-0.66	-1.1	3.6	7.8
3	6.7	0.0	0.0	0.0	0.0

Clustering and locally fitting to quadratic functions The next step is locally fitting quadratic value functions to the closed-loop data. The local Q function takes the form of

$$Q(x, u) = \theta^T \Phi(x, u) \tag{22}$$

where $\Phi(x, u)$ is the regressor vector of

$$[x_1^2,\ x_2^2,\ u^2,\ x_1x_2,\ x_1u,\ x_2u,\ x_1,\ x_2,\ u,\ 1]^T \tag{23}$$

The cluster containing equilibrium point (origin) is fitted using only the first 6 terms of the above vector. This ensures that the minimum value is zero at equilibrium point regardless of fitting error. The local data sets for identifying clusters in the state-action-parameter space were constructed using 49 neighboring points. The state vector was multiplied by 50 for better scaling. It was found that three clusters yielded the most distinctive clusters for the fitting.

Improving the initial policy We improve the initial value function obtained from PID controllers by selecting a greedy control action at each time as in (7). The observed value associated with a greedy control action is then used to update the value function.

(1) Choose $(\alpha, \gamma, \lambda) = (0.99, 10^{-3}, 1)$
(2) Select the initial state point from a batch.
(3) Repeat the following steps within each episode
  (a) Given $x(t) = [x_1(t)\ x_2(t)]^T$ at time t, determine the range of u to optimize over. The range of u is the same as that of fitted data. This avoids undue extrapolation in the absence of a process model. SVM is used to determine clusters corresponding to $x(t),\ u$ pairs.
  (b) The greedy action is then calculated by
$$u(t) = \arg\min_{u(t)} \tilde{Q}_t(x(t), u(t)) \tag{24}$$
  (c) Observe stage-wise cost ϕ and the successor state $x(t+1)$, and calculate the one-step TD error
$$\delta(t) = \phi(x(t), u(t)) + \alpha \min_{u(t+1)} \tilde{Q}_t(x(t+1), u(t+1)) - \tilde{Q}_t(x(t), u(t)) \tag{25}$$
  (d) Update the θ according to
$$\theta(t+1) = \theta(t) + \gamma\delta(t)\nabla_\theta Q_t e_t \tag{26}$$
(4) $\gamma(i+1) = \gamma(i) \times 0.9^i$ where i is the batch number.

Table 1 shows values of θ after 50 batches, and Figure 1 shows a plot of $\tilde{J}(x)$ obtained by optimizing $\tilde{Q}(x, u)$ over u after 50 batches (1250 episodes).

On-line performance of a learned control policy Figure 2 shows an on-line simulation of the system under the initial PI controllers and the improved value-function-

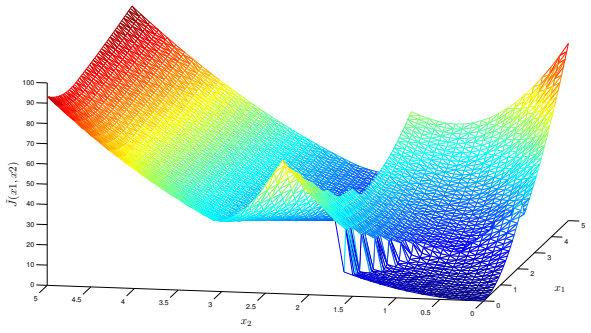

Fig. 1. Plot of parameterized $\tilde{J}(x)$ after 50 batches

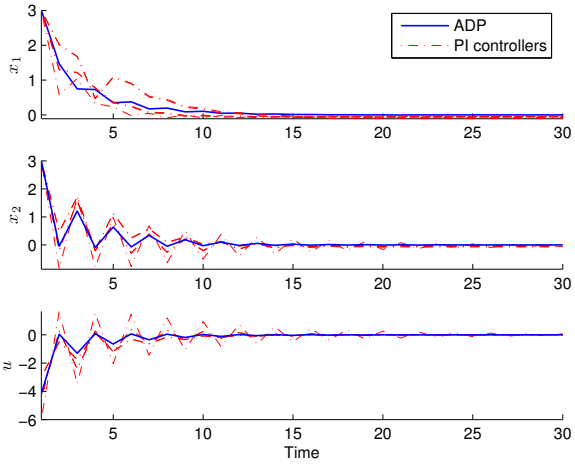

Fig. 2. A sample result of on-line implementation under initial and learned control policies

Table 2. Hybrid system example: Comparison of on-line performances of control policies : SSE is the average value taken over one batch (25 trajectories).

Policy	Sum of Squared Error (Batch)
$(K_c,\ \tau_I) = (1,\ 10)$	760.0
$(K_c,\ \tau_I) = (1,\ 15)$	759.6
$(K_c,\ \tau_I) = (1.5,\ 10)$	1339.0
$(K_c,\ \tau_I) = (1.5,\ 15)$	1341.5
$(K_c,\ \tau_I) = (2,\ 10)$	3134.6
$\tilde{Q}$	289.4

based controller. Regulation performance in terms of one-batch SSE is shown in Table 2. It is evident that the learned policy shows a significant improvement over the initial policy even in the absence of a process model.

4.2 Nonlinear System

We consider a nonlinear bioreactor example in Bequette [2003]. The bioreactor model is given by

$$\begin{aligned} \frac{dx_1}{dt} &= (\mu - D)x_1 \\ \frac{dx_2}{dt} &= D(x_{2f} - x_2) - \frac{\mu x_1}{Y} \\ y &= x_1 \end{aligned} \tag{27}$$

Table 3. Bioreactor example: Tuning parameters of the IMC controllers

PI Controller	K_c	τ_I	λ
1	5.86	5.89	2.06
2	5.86	5.89	2.18
3	5.86	5.89	2.36
4	5.86	5.89	2.53
5	5.86	5.89	2.65

Table 4. Learned parameters of the value function for the bioreactor example. The upper bar denotes deviation variables from the equilibrium point.

Cluster	$\bar{x}_1^2$	$\bar{x}_2^2$	$\bar{u}^2$	$\bar{x}_1\bar{x}_2$	$\bar{x}_1\bar{u}$
1	1.4802	-0.2642	-0.3835	0.8163	0.4366
2	1.0351	-0.0095	0.3461	-0.3721	-1.4061
3	1.4162	-0.2282	0.6141	0.0605	-2.0307
4	0.1318	-0.0756	-2.4243	-0.0676	2.6461
5	5.7797	0.2115	3.1445	0.0894	-6.3042
Cluster	$\bar{x}_2\bar{u}$	$\bar{x}_1$	$\bar{x}_2$	$\bar{u}$	1
1	-0.9177	-3.3870	1.1573	3.2273	-0.8141
2	0.2525	2.0951	0.2516	-0.1375	-0.6082
3	-0.1136	1.6646	1.0690	0.2731	-1.7157
4	-0.0376	-1.5490	0.3547	2.6718	-0.3381
5	0.3174	0.0000	0.0000	0.0000	0.0000

where x_1 and x_2 are the concentrations of biomass and substrate, respectively. The dilution rate D is the manipulated variable. x_{2f} and μ are substrate feed concentration and specific growth rate. The substrate inhibition model is used to define the specific growth rate

$$\mu = \frac{\mu_{\max} x_2}{k_m + x_2 + k_1 x_2^2} \tag{28}$$

where $\mu_{\max} = 0.53hr^{-1}$, $k_m = 0.12g/L$, $k_1 = 0.4545L/g$, and $Y = 0.4$.

The objective is to maintain the system at the unstable equilibrium point of $(x_1, x_2, D) = (0.995, 1.512, 0.3)$. First, closed-loop simulations were performed with five different discrete PI controllers with the sample time of 0.1 (See Table 3). The tuning parameters were selected based on the IMC tuning rule described in Bequette [2003]. We randomly sampled initial state points from $-5 \le x_1, x_2 \le 5$ and collected 896 data points to identify clusters and approximate the value function.

The stage-wise cost was defined as $\phi(x(k), u(k)) = y(k+1)^2$ with the discount factor of 0.99, and piecewise affine quadratic functions were fitted to the state-action pair and their corresponding values based on the closed-loop data under the five PI controllers. Five clusters were chosen as in the hybrid system example with the multiplication factor for the state being 100. The learned parameters are shown in Table 4.

The gradient-based on-line updating was performed with the initial learning parameter $\gamma = 10^{-6}$. The learned value function could improve the initial PI control policies by combining their good features. Fig. 3 shows state and input trajectories for the IMC controllers and the suggested ADP approach. The performance comparison over one batch in Table 5 shows that the ADP could improve the control performance starting from PI controllers without any process model.

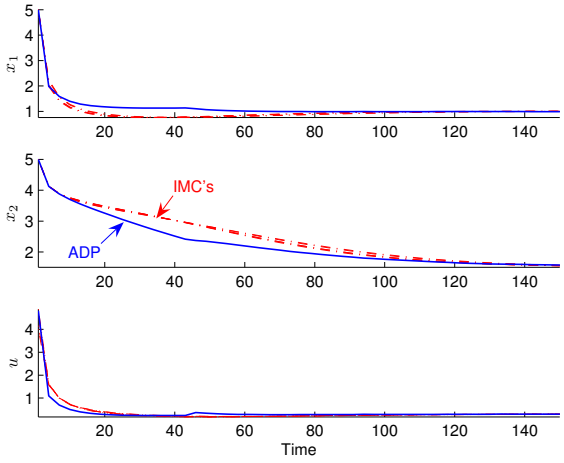

Fig. 3. Bioreactor example: state and input trajectories

Table 5. Bioreactor example: Comparison of on-line performances (SSE) of control policies for one batch

IMC 1	IMC 2	IMC 3	IMC 4	IMC 5	ADP
239.6	249.8	267.7	288.6	350.4	222.5

5. CONCLUSIONS AND FUTURE WORKS

In this paper, we presented a novel ADP based controller that is suitable for complex systems with continuous state and control action spaces. A piecewise quadratic representation is used to reduce the computational and data storage requirements. The local convexity of the approximation also facilitates the optimization step. In constructing local quadratic value functions, data clustering is performed in the state-action-parameter space to locate the centroids based on the local structure of value functions. The optimal number of clusters was chosen by considering fitting performance and degree of distinction between clusters. This is mainly affected by size of LDs and weights given to parameters and states in the augmented state-action-parameter vector. Quantitative selection of optimal number of clusters is under investigation.

The convergence of the proposed algorithm is still an open issue. We found that the convergence depends on the learning rate γ, the "degree of exploration," and number and location of clusters. However, using a relatively "cautious" learning rate and limiting the range of control action could provide stability with performance improvement.

ACKNOWLEDGEMENTS

We thank the University of Alberta and the Natural Sciences and Engineering Research Council of Canada (NSERC) for financial support.

REFERENCES

R. E. Bellman. *Dynamic programming*. Princeton University Press, 1957.

B. W. Bequette. *Process Control: Modeling, Design, and Simulation*. Prentice Hall, 2003.

D. P. Bertsekas. *Dynamic programming and optimal control*, volume I. Athena Scientific, 2nd edition, 2000.

G. Ferrari-Trecate, M. Muselli, D. Liberati, and M. Morari. A clustering technique for the identification of piecewise affine systems. *Automatica*, 39:205–217, 2003.

G. J. Gordon. Stable function approximation in dynamic programming. Technical Report CMU-CS-95-103, School of Computer Science, Carnegie Mellon University, 1995.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer-Verlag, 2001.

J. H. Lee and Z. Yu. Worst-case formulations of model predictive control for systems with bounded parameters. *Automatica*, 33(5):763–781, 1997.

J. M. Lee and J. H. Lee. Approximate dynamic programming-based approaches for input-output data-driven control of nonlinear processes. *Automatica*, 41: 1281–1288, 2005.

J. M. Lee, N. S. Kaisare, and J. H. Lee. Choice of approximator and design of penalty function for an approximate dynamic programming based control approach. *Journal of Process Control*, 16:135–156, 2006.

J. Morris. New horizons for process control in chemical and process engineering. *Trans IChemE*, 81(Part A): 199–200, February 2003.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: an introduction*. A Bradford book, 1998.

V. Vapnik. *Statistical learning theory*. Wiley, 1998.

C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.