Approximate dynamic programming approach for process control

Jay H. Lee * **Wee Chin Wong** *

* *School of Chemical & Biomolecular Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: jay.lee,wwong@chbe.gatech.edu).*

Abstract: In this talk, we assess the potentials of the approximate dynamic programming (ADP) approach for process control, especially as a method to complement the model predictive control (MPC) approach. In the Artificial Intelligence (AI) and Operations Research (OR) research communities, ADP has recently seen significant activities as an effective method for solving Markov Decision Process (MDP), which represents a type of multi-stage decision problems under uncertainty. Process control problems are similar to MDPs with the key difference being the *continuous* state and action spaces as opposed to discrete ones. In addition, unlike in other popular ADP application areas like robotics or games, in process control applications first and foremost concern should be on the safety and economics of the on-going operation rather than on efficient learning. We explore different options within ADP design, such as the pre-decision state vs. post-decision state value function, parametric vs. nonparametric value function approximator, batch-mode vs. continuous-mode learning, exploration vs. robustness, etc. We argue that ADP possesses great potentials, especially for obtaining effective control policies for stochastic constrained nonlinear or linear systems and continually improving them towards optimality.

Keywords: Stochastic optimal control, constraints

1. INTRODUCTION

Model predictive control (MPC) is a technique in which the current control action is obtained by minimizing on-line, a cost criterion defined on a finite time interval. Nominal deterministic trajectories of future disturbance signals and uncertainties are necessarily assumed in order to obtain an optimization problem amenable to on-line solution via math programming. The solution generates a control sequence from which the first element is extracted and implemented. The procedure is repeated at the next time instant. Owing to its ability to handle constrained, multi-variable control problems in an optimal manner, MPC has become the de-facto advanced process control solution for the process industries today.

MPC is by now considered to be a mature technology owing to the plethora of research and industrial experiences during the past three decades. Despite this, it has some fundamental limitations, which prevents it from being a panacea for all process control problems. One well-known limitation is the potentially exorbitant on-line computation required for solving a large-scale, and potentially non-convex math program that scales with the dimension of the state as well as the length of prediction horizon. Recent developments (Laird and Biegler (2008)) have made some headway in tackling this problem although nontrivial computational challenges still exist.

⋆ JHL gratefully acknowledges financial support from Owens Corning and Weyerhaeuser.

The second limitation arises from the fact that the deterministic formulation adopted by MPC is inherently incapable of addressing uncertainty in a closed-loop optimal fashion. Its open-loop optimal control formulation used to find the control moves at each sample time means the fact that information about future uncertainty will be revealed, this being generally beneficial for control performance, is not considered. Most of the past attempts at ameliorating the impact of uncertainty has been reflected in robust MPCs formulations based on the objective of minimizing the worst-case scenarios (Scokaert and Mayne (1998)) at the expense of overly conservative policies. Multi-scenario formulations (Laird and Biegler (2008)) have also been developed but the number of scenarios is limited and they do not give closed-loop optimal policies in general. Stochastic programming based methodologies (Pena et al. (2005)) allow for recourse actions at the computational expense of enumerating an exponentially growing number of scenarios.

In this paper, we examine the possibility of lessening or removing the above-mentioned limitations by combining MPC with an approach called "approximate dynamic programming (ADP)." ADP is a technique that surfaced from the research on reinforcement learning in the Artificial Intelligence (AI) community (Sutton and Barto (1998); Bertsekas and Tsitsiklis (1996)). It has its theoretical foundations in the traditional dynamic programming by Richard Bellman (Bellman (1957)) but its computational bottlenecks, termed as "the curse of dimensionality" by Bellman himself, are relieved through ideas such as intelli-

gent sampling of the state space through simulations and function approximation. ADP, due to its root in AI, has mainly been studied in the context of Markov Decision Processes (MDPs), which involve discrete finite state/ action spaces and probabilistic transitions. Hence, its application to process control problems, which typically involve continuous state/ action spaces, is not straightforward. In addition, the characteristics of process control problems are somewhat different from those of robotics, games, and resource allocation problems. For example, in process control applications, the idea of "learning by mistakes" for the sake of efficient learning, may not be tolerated as mistakes often bring unacceptable consequences in terms of safety and economics. Hence, extension of ADP to process control may require significant care and possibly some new tools.

Design of an ADP algorithm involves a variety of choices, including type of function approximator, pre-decision vs. post-decision formulation, batch vs. continuous updating of the value table, and exploration vs. robustness tradeoff. We will visit these issues, carefully examining the implications of these choices in the context of designing a learning algorithm for process control applications. In addition, we will also consider the complementary nature or synergies between ADP and MPC.

The rest of the paper is organized as follows. In Section 2, we will briefly review the basics of MDP, ADP and also present a mathematical representation of the system we consider for control. In Section 3, we will examine the various options and choices and their implications for process control applications. In Section 4, we will present a few examples, including those involving both linear and nonlinear stochastic systems. In Section 5, we conclude the paper and discuss other control-related areas where ADP can potentially be useful in the process industries.

2. BACKGROUND

2.1 Markov Decision Processes and Approximate Dynamic Programming

Markov Decision Processes (MDPs) provide a framework for modeling real world processes that have a stage-wise structure. The stage can denote a time epoch or other quantities like location, processing step, etc. At any stage, the system is recognized as being in a state (designated as s), which is a set of attributes that aid decision-making. The set of all possible states is called state space (designated as S). Starting in state s belonging to S, there is a set of actions from which the decision-maker must choose. The set of all possible actions is called action space (A) and an element of the action space is denoted by a. When action a is taken in state s, and the system transitions to the next stage, it ends up in a unique next state $s' \in S$ in the absence of any uncertainty. However, for stochastic problems, there is a set of possible next states for each state-action pair. The probability of transition to a particular next state in this case is governed by a state transition probability function, P. In the process, reward $r(s, a, s')$ is received, which is determined by the reward function r. The dependence of r on s' is often suppressed by taking a weighted average over all possible states at the next stage. At each stage, actions are taken so that the sum of stage-wise rewards is maximized. In the presence of uncertainty, the expected sum of rewards is maximized. When infinite stages are present, i.e., extremely large time horizon, the future rewards are often discounted using a discount factor γ. When the number of stages is infinite, the problem is called an infinite horizon MDP as opposed to a finite horizon MDP for finite number of stages. In most applications, a stage symbolizes a time epoch. Therefore, the term time epoch or time step is often used synonymously with 'stage'.

More formally, MDP is defined by a tuple (S, A, P, R, γ) where S is a set of states, A is a set of actions, $P : S \times A \times S \rightarrow [0, 1]$ is a set of transition probabilities that describe the dynamic behavior of the modeled environment, $R : S \times A \times S \rightarrow \mathbb{R}$ denotes a reward model that determines the stage-wise reward when action a is taken in state s leading to next state s' and $\gamma \in [0, 1)$ is the discount factor used to discount future rewards. A γ value close to 0, places very little weightage on future rewards, while γ close to 1 results in very little discounting.

One of the fundamental properties of the MDPs is that the transition and reward functions associated with the stage-wise transition of state are independent of the past states and actions. Referred to as Markov property, this memory-less feature enables the decomposition of the overall optimization problem into separate stage-wise problems. This is accomplished by using a recursive relationship between the value of being in a state at any stage.

An important notion in this regard is the so called value function denoted by $V(s)$, which is defined as the (often discounted) sum of rewards over a time horizon which can be either finite or infinite (shown below) and discussed hereafter:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, \mu(s_t)) | s_0 = s\right] \tag{1}$$

where t denotes the time epoch, s_t is the state at time t and $\pi : S \rightarrow A$, is the policy that dictates the choice of action for a given state at time t.

The goal is to find an optimal policy that maximizes the value function for all $s \in S$. This is achieved by solving the Bellman equation (Bellman (1957)) for finite or infinite horizon problems. The optimal policy can be derived via dynamic programming. Let $a^*(s)$ be the optimal action to be taken when the system is in state s, independent of time t. $V^*(s)$ is called the optimal value function and is obtained as the solution to the (Bellman equation) (2), which must be solved for all $s \in S$:

$$V^*(s) = \max_{a \in A} \left\{ r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) V^*(s') \right\} \quad \forall s \tag{2}$$

$$a^*(s) = \pi^*(s) \triangleq \arg\max_{a \in A} \left\{ r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) V^*(s') \right\} \tag{3}$$

where symbol $p(.)$ denotes the probability of a quantity. It is well-known (Putterman (1994)) that for infinite horizon problems, a stationary optimal policy of the form in (3) exists, where $V^*(s)$ is the average discounted infinite horizon reward obtained when the optimal policy is followed starting from s until infinity (Putterman (1994)). This implies

that the state to action mapping in the form of optimal policy is independent of the time epoch. The existence of stationary optimal policy is conditioned on the properties of model elements. One of the sufficient conditions is that there be a finite action space A_s corresponding to each state $s \in S$, maximum attainable stage-wise reward is finite and discount factor $\gamma \in [0, 1)$. The alternative sets of sufficient conditions for existence of a stationary optimal policy for discounted infinite horizon MDPs can be found in (Putterman (1994)).

It must be noted that the set of Bellman equations also called optimality equations are difficult to solve analytically because of the presence of the max operator. One of the popular solution methods is called value iteration (Putterman (1994)): Starting with an arbitrary value function $V_0(s)$ for each state $s \in S$, the value function is iteratively improved by successive substitution into (2) until ϵ-convergence is reached. The operator for one iteration, that is the maximization in (2), can be denoted as H such that $V_{n+1} = HV_n$. The sequence of estimates of value function $V(s)$, $\forall s \in S$, converges to a fixed point solution. This is a consequence of Banach's theorem for contraction mappings (Putterman (1994)). Since H is a proven contraction map, the convergence properties hold.

Due to ease of implementation, value iteration is perhaps the most widely used algorithm in dynamic programming. Certain other methods like policy iteration (Bertsekas (2005)), a hybrid between value iteration and policy iteration (Powell (2007)) and linear programming method for dynamic programs (Farias and Roy (2003)) are also used depending on the problem structure. The complexity of the value-iteration algorithm grows as a function of $o(|S|^2 \times |A|)$. This is attributed to the following three aspects of the value iteration:

(1) Equation (2) needs to be solved for all s belong to S, so the solution time is directly proportional to $|S|$.
(2) The complexity of max operation depends on the size of the action space $|A|$.
(3) The calculation of expectation within the max operator depends on the number of possible next states, i.e., $|S|$.

In the presence of very large state and (or) action spaces, the value iteration algorithm cannot be implemented in its exact form. Several approximation methods have been developed to circumvent this difficulty, including: approximate dynamic programming methods using value function approximations (Powell (2007)), Q-learning, temporal difference learning (Sutton and Barto (1998)) functions (Farias and Roy (2003)) and dynamic programming methods using post decisions state (Powell (2007)).

All the above methods assume that the system state is completely known or observed at all times. When this assumption does not hold, the equivalent framework is called a Partially Observed Markov Decision Process (POMDP) (Cassandra et al. (1994)), for which a significant but less body of literature exists.

2.2 System Definition: Process Control Problems

Consider the optimal control of the following discrete-time stochastic system:

$$x_{t+1} = f(x_t, u_t, \omega_t) \tag{4}$$

where $x_t \in \mathcal{X} \subseteq \mathbb{R}^{n_x}$ refers to the system state at discrete time index t, $u_t \in \mathcal{U} \subseteq \mathbb{R}^{n_u}$ a control or action vector, and ω_t an exogenous, unmeasured, stochastic signal. x may contain physically meaningful states as well as measured disturbances, and parameters subject to uncertainty. f refers to the single-stage transition function. For problems where the system's dynamics are represented by ordinary differential equations, f is then the result of numerical integration across a single sample-time, with vectors u and ω held constant. Throughout this paper, it is assumed that full state feedback is available. In the event that only output feedback is available, x is interpreted as an information vector that contains the sufficient statistics of the state estimate's probability density function. Such lifting is possible as the information vector is governed by another related set of differential equations. (*i.e.*, the filter dynamics).

Let $\mu \in \Gamma$ be a 'state-feedback policy' that maps the state vector to the action vector, where Γ represents the set of all admissible (stationary) such policies. To distinguish from the earlier value function $V(s)$, $J^\mu(x)$ will be used to denote the 'cost-to-go' function, which is defined as the infinite horizon, discounted sum of the stage-wise costs under the policy μ starting from an arbitrary state x:

$$J^\mu(x) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k \phi(x_k, u_k = \mu(x_k)) | x_0 = x\right] \tag{5}$$

where ϕ represents a pre-specified stage-wise cost (*e.g.* $\phi(x, u) := ||x||_Q^2 + ||x||_R^2$) and $\gamma \in [0, 1)$ is a discount factor. The goal then is to find the optimal (stationary) policy $\mu^* : \mathcal{X} \to \mathcal{U}$, that yields the minimum cost-to-go function as below:

$$J^{\mu^*}(x) = \min_{\mu \in \Gamma} \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k \phi(x_{t+k}, u_{t+k} = \mu(x_{t+k})) | x_t = x\right] \tag{6}$$

$J^{\mu^*} : \mathcal{X} \to \mathbb{R}^{0+}$ is the *optimal* 'cost-to-go' function and is an indication of the attractiveness of a given state in terms of future rewards. By definition, $J^{\mu^*}(x) \leq J^\mu(x), \forall x$ and $\forall \mu \in \Gamma$.

The main difference between the above and the previously introduced MDPs is that the state and action spaces are continuous. However, the fundamental concepts of DP still apply here. Based on the principle of optimality (Bellman (1957)), one is able to re-write (6), thereby obtaining Bellman's optimality equations:

$$\begin{aligned} J^{\mu^*}(x) &= \min_{u \in \mathcal{U}} \left\{\phi(x, u) + \gamma \mathbb{E}_{(\omega|x)}[J^{\mu^*}(f(x, u, \omega))]\right\} \\ &= \left(T J^{\mu^*}\right)(x) \end{aligned} \tag{7}$$

T above represents the single-pass DP operator represented by the minimization operation. The optimal policy is implicitly obtained through the solution of the associated single-stage optimization:

$$\mu^*(x) = \arg\min_{u \in \mathcal{U}} \left\{\phi(x, u) + \gamma \mathbb{E}_{(\omega|x)}[J^{\mu^*}(f(x, u, \omega))]\right\} \tag{8}$$

In principle, the optimal control problem is solved once J^{μ^*} is known. It is noted that for deterministic problems (where the expectation operator is dropped), the DP formalism provides a convenient way of solving multi-stage problems through an equivalent single-stage optimization.

Unfortunately, analytical solutions to Bellman's optimality equations are available for only a small class of problems, of which the celebrated Linear Quadratic Gaussian (LQG) problem is one. For situations of practical interest, numerical techniques are required. Similar to the case of discrete state and action space, the repeated application of T on an arbitrarily initialized cost-to-go leads to convergence and underpins the idea behind Value Iteration (VI).

$$J^{\mu^*}(x) = TJ^{\mu^*}(x) = \lim_{i\to\infty}(T)^i J^{\mu}(x),\ \forall \mu, x \qquad (9)$$

In process control problems, due to the continuous nature of the state and action spaces which must be discretized, numerical solutions become quickly bottle-necked as the problem dimensions grow. In fact, the growth would be exponential as the number of discretized points grows with the dimension as such. Hence, a naive application of VI in this case is computationally prohibitive and the 'curse-of-dimensionality' is even more apparent in continuous problems. For problems with continuous state and action space, one needs to resort to approximations that involve an intelligent state-sampling/ discretization scheme and/ or an efficient representation of the cost-to-go (Lee and Lee (2006); Powell (2007)).

2.3 Approximate dynamic programming for problems with continuous state and action space

Value iteration or policy iteration in general can work with only finite state space. For systems with continuous state and action space, one must then work with discretized state state, either through gridding, or more preferably, sampling. It is often the case that only a small portion of $\mathcal{X}$ and $\mathcal{U}$ will ever be visited under optimal and/ or high-quality sub-optimal policies. This is especially true when the dimension of the state space is large compared to that of the input. Let us denote the subset of the state space that is 'relevant', *i.e.*, visited with non-trivial probability under the optimal control, as $\mathcal{X}^*_{REL}$. Such a set would be continuous but much smaller-sized than $\mathcal{X}$ in general. The key notion is that if one could identify $\mathcal{X}^*_{REL}$ or a parsimonious superset of it, one can sample the set with sufficient density to perform the dynamic programming at significantly reduced computation. Of course, the difficulty is that it is not easy to obtain such a set ahead of time without knowing the optimal controller itself.

The ADP approach proposed by the authors of (Lee and Lee (2004, 2006); Tosukhowong and Lee (2009)) for process control applications, the skeleton of which is described in this subsection, employs carefully designed simulation schemes for the sampling of the state space and function approximation (for the purpose of cost-to-go interpolation) to this end. For the VI-variant, we have the following off-line computations:

(1) Identify a finite-sized, 'relevant' state-space, $X_{sam} \subset \mathcal{X}$, $|X_{sam}| = N$. This is achieved, for instance, by simulating all possible combinations of sub-optimal policies (potentially with dithering) and operating conditions. The latter are defined as all starting states of interest (for servo problems) as well as potential values of measured disturbance values. Dithering may also be introduced for the purpose of exploration.
(2) Assign a cost-to-go for all elements of X_{sam}, using the simulation data according to (5). The initial, finite-sized 'cost-to-go' table, denoted by $\mathcal{T}_{[0]} \triangleq \{x, \hat{J}^{\mu^*}_{[0]}(x) \,|x \in X_{sam}\}$, is obtained. The symbol $\hat{(\cdot)}$ is used to emphasize the approximate nature of the cost-to-go sequence, even at its limit. Exact initialization is not critical per se since the fixed point derived from the following step is unique.
(3) Obtain converged cost-to-go values for X_{sam} through VI, yielding the sequence of value tables $\{\mathcal{T}_{[0]}, \mathcal{T}_{[1]}, \ldots\}$. Since the VI requires the evaluation of the cost-to-go function for states $(f(x,u,\omega))$ not necessarily in X_{sam}, a well-designed function approximator is needed to interpolate among the stored values (see discussion in Section 3.1). A certain choice of function approximator ensures that each pass of the iteration is a *contraction-map* with a unique fixed point (see Section 3.1). In other words, each step of the modified VI involves:
$$\hat{J}^{\mu^*}_{[i+1]}(x) = \left(TF(\hat{J}^{\mu^*}_{[i]})\right)(x),\ \forall x \in X_{sam} \qquad (10)$$
Here $F(\hat{J}^{\mu^*})$ denotes the cost-to-go function approximator based on the stored values $\{\hat{J}^{\mu^*}_{[i]}(x), x \in X_{sam}\}$. Termination occurs when $\|\hat{J}^{\mu^*}_{[i+1]} - \hat{J}^{\mu^*}_{[i]}\|_\infty$ is less than a pre-defined tolerance.
(4) Return to step 1, since the relevant domain of the state-space may not be properly ascertained a-priori. Otherwise, use the converged values for online control.

The authors of Ma and Powell (2009) used an approximate policy iteration scheme where $J^{\mu}(x), \forall x \in \mathcal{X}$ is assumed to be linear in a set of basis functions (known or otherwise assumed to be orthogonal polynomials of sufficiently large degree). The coefficients are learnt through a least-squares procedure once the system of interest is allowed to evolve according to the current policy, which is similar to step (1) where relevant states are collected. The limitation is that suitable basis functions are difficult to ascertain in general.

3. ISSUES AND CHOICES

3.1 Function approximation and stable learning

The need for function approximation for the purpose of generalization has been discussed. Given a training set $\mathcal{T} \triangleq \{x_i, \hat{J}(x_i)\}_{i=1}^N$, a value table composed of a finite number (N) of input $(x_i \in \mathcal{X})$ and target values $(\hat{J}(x_i) \in \mathbb{R})$, a function approximator, F, whose domain is $\mathcal{X}$, maps a query point $x_q \in \mathcal{X}$ to a subset of the real line.

The dominant and natural choice for function approximators has typically involved parametric global approximators such as neural networks or the use of basis functions

such as high order orthogonal polynomials or Fourier series (Tsitsiklis and Roy (1996); Konidaris and Osentoski (2008)). While this approach has met with some success in certain applications (*e.g.* in Backgammon (Tesauro (1992)), it is not immune from divergent behavior (Lee and Lee (2004, 2006)) when employed in the context of ADP. In certain cases, the off-line iteration would fail to converge, with the cost-to-go approximation showing non-monotonic behavior or instability with respect to iterations. Thrun and Schwartz (1993) were the first to attribute the failure with function approximation to an 'over-estimation' effect. Sabes (1993) demonstrated that sub-optimality can be severe when a global approximator with a linear combination of basis functions is employed. Boyan and Moore (1995) provide insightful illustrations showing the failure of popular function approximators during off-line learning.

There are considerably fewer papers that address function approximation schemes for problems with continuous state and action spaces (Ma and Powell (2009), Lee et al. (2006)). The problem of linear quadratic regulation, for which the value function is known to be quadratic in structure, is a noted exception Bradtke (1993). Ormoneit and Sen (2002) proposed a kernel-based approach for problems with continuous states but finite actions and demonstrate convergence to the optimal cost-to-go value function with an increasing number of samples and decreasing kernel bandwidth under a model-free scheme. Ma and Powell (2009) proposed a provably convergent approximate policy iteration under the assumption of known basis functions and other technical conditions.

Stable learning during the off-linear value iteration step of the proposed ADP strategy is highly desirable as it can be frustrating to run a large number of iterations only to have the result "blow up" all of sudden due to some complicated coupling between the function approximation error and value iteration. To have provable convergence of the approximate value iteration (not necessarily to the optimal value function, however), one needs to use a function approximator with a certain property called "non-expansion" . Gordon (1995) discussed the viability of using such a class of function approximators. With such a choice, the overall operator composed of value-iteration and then function approximation results can be shown to be a contraction map therefore ensuring convergence.

Definition 1. A γ-contraction mapping m defined on a normed vector space (mapping elements from this space, $\mathcal{V}$, to itself) is defined as such:

$\forall v_1, v_2 \in \mathcal{V}$, $||m(v_1) - m(v_2)|| \leq \gamma ||v_1 - v_2||$, $\gamma \in [0, 1)$

where v_1, v_2 are arbitrarily chosen elements of $\mathcal{V}$.

Definition 2. When $\gamma = 1$, $m : \mathcal{V} \to \mathcal{V}$ is termed a non-expansion (Gordon (1995)).

From Banach's fixed-point theorem, it can be easily shown that every the iterated sequence $\{v, m(v), m^2(v), \ldots\}$ converges to a unique fixed point. As explained earlier, the proposed ADP method starts with initial estimates $\hat{J}^{\mu^*}_{[0]}(x)$, $\forall x \in X_{sam}$. This is followed by function approximation (recall that this mapping is denoted by F), and an application of the DP operator, T to yield $\hat{J}^{\mu^*}_{[1]}$. The process is repeated again. Our experience with the ADP approach (Lee and Lee (2004, 2005)) has been that stability of learning and the quality of a learned control policy are critically dependent on the structure of the function approximator. A sufficient condition for convergence is to demonstrate that the overall operator T with function approximator F is a contraction map. This, in turn, holds true if F is a non-expansion map.

Proposition 1. T is a γ-contraction map if F is non-expansive.

Proof. Given arbitrary vectors $\hat{J}_1, \hat{J}_2 \in \mathbb{R}^N$,

$$||TF(\hat{J}_1) - T^{\mu^*}F(\hat{J}_2)||_\infty \leq \gamma ||F(\hat{J}_1) - F(\hat{J}_2)||_\infty \quad (11)$$

$$\leq \gamma ||\hat{J}_1 - \hat{J}_2||_\infty \quad (12)$$

The first line is true since T is a γ-contraction map defined on the space of value functions. The second inequality follows if one employs a function approximator with a non-expansion property.

Function approximators that employ averaging, as defined below, can be shown to possesses a non-expansion property.

Definition 3. F is an averager if every fitted valued is the weighted average of of target values, potentially with the addition of a bias term. Specifically,

$$F(\hat{J})(x_q) = \beta_0(x_q) + \sum_{i=1}^{N} \beta_i(x_q)\hat{J}(x_i) \quad (13)$$

Here, $\{\beta_i\}_{i=0}^N \geq 0$, and $\sum_{i=1}^{N} \beta_i \leq 1$. Note that the weights β are allowed to depend on the query point (x_q) and input values $(\{x_i\}_{i=1}^N)$ but not the target values.

That such an averager is a non-expansion (*i.e.* (12) is true) is easily demonstrated.

One such type of approximator we have experimented with extensively is instance-based (Lee et al. (2006)) local averagers, such as k-Nearest Neighbors-based (kNN) predictors. Instance-based algorithms are non-parametric representations using stored points 'close' to a query point for making predictions. Closeness is usually defined according to some distance metric (such as Euclidean distance). Predictions of the weighted kNN are given by:

$$F(\hat{J})(x_q) = \beta_0(x_q) + \sum_{x_i \in \mathcal{N}_k(x_q)} \beta_i(x_q)\hat{J}(x_i) \quad (14)$$

where $\mathcal{N}_k(x_q)$ refers to the set containing the k points closest to x_q. The weights (normalized by constant c) are defined as: $\beta_i = c((x_q - x_i)^T W (x_q - x_i))^{-0.5}$, $i \geq 1$. W is a feature weighting matrix use to scale and also to emphasize dimensions that are more important.

3.2 Cautious learning for robustness

It has been demonstrated (Smart and Kaelbling (2000); Lee et al. (2006)) that simply using a local averager (with $\beta_0 = 0$), though guaranteeing convergence, does not necessarily give a converged function leading to a stable closed-loop behavior. This is because function approximation

error can be significant, particularly when the training data is insufficient. Safeguards against 'over-extrapolation' during value iteration is often needed for the successful implementation of the proposed ADP method. For a query point located in regions with little data present, distance-weighted averaging may fail to provide meaningful generalizations of the cost-to-go. Prevention of taking such a query point may be achieved by including in the cost-to-go term β_0, a penalty that is imposed whenever the minimization step encounters a query point (x_q) far away from X_{sam}:

$$\beta_0(x_q) = A.U\left(\frac{1}{f_\Omega(x_q)} - \rho\right) \cdot \left(\frac{\frac{1}{f_\Omega(x_q)} - \rho}{\rho}\right)^2 \quad (15)$$

Here, ρ is a data-density threshold value, A a scaling parameter, and U, the Heaviside step function that returns a zero value whenever its argument is non-positive and unity, otherwise. $f_\Omega(x_q)$ is a measure of data density as ascertained by fitting a Kernel density estimator over training set Ω:

$$f_\Omega(x_q) = \frac{1}{N_\Omega}\sum_{i}^{N_\Omega} K\left(\frac{x_q - x_i}{\sigma}\right) \quad (16)$$

where kernel $K(\cdot)$ refers to a zero-mean Gaussian with variance $\sigma^2 I_{n_x}$. For generality, Ω is allowed to differ from X_{sam}. Furthermore, a bound is imposed on β_0 whenever it exceeds a threshold value. Tuning rules for ρ, A, σ can be found in (Lee et al. (2006)) and are not reproduced here in the interest of brevity.

3.3 Pre-decision vs. post-decision state formulation

For stochastic control problems, the single-stage optimization required during off-line value-iteration (see (7)) and on-line implementation of the optimal policy (see (8)) requires the generally cumbersome evaluation of an expectation.

The use of an intermediate post-decision (x^p) state, first introduced by Roy et al. (1997) and employed extensively by Powell (2007) in solving operations research problems, oftentimes allows for more computationally effective strategies. x^p refers to the the system state immediately after the control vector is introduced but before the uncertainty is realized. As a result, f is decomposed into the following sub-transitions:

$$x_t^p = f_1(x_t, u_t)$$
$$x_{t+1} = f_2(x_t^p, \omega_t)$$

where the composition of f_1 and f_2 is equivalent, in effect, to f, in (4). Note that f_1 describes a deterministic transition between the pre-decision state variable (x) and x^p. f_2 involves the transition due to uncertainty after the control action is implemented. Consequently, the value function of x^p, $J^{\mu,p}(x^p)$, may be expressed in terms of the value function of x, as such:

$$J^{\mu,p}(x_t^p) = \mathbb{E}_{(\omega|x_t^p)}\left[J^\mu(x_{t+1})\right], \forall \mu \quad (17)$$

By considering the optimal policy μ^*, and substituting (7) into (17), the min and $\mathbb{E}$ operators are interchanged, yielding:

$$J^{\mu^*,p}(x_t^p) = \mathbb{E}_{(\omega|x_t^p)}\left[\min_{u_{t+1}\in\mathcal{U}}\left\{\phi(x_{t+1}, u_{t+1}) + \gamma J^{\mu^*,p}(x_{t+1}^p)\right\}\right] \quad (18)$$

The single-stage on-line optimization is also streamlined:

$$\mu^*(x) = \arg\min_{u\in\mathcal{U}}\left\{\phi(x,u) + \gamma J^{\mu^*,p}(f_1(x,u))\right\} \quad (19)$$

The introduction of the post-decision state allows the generally non-commutative min and $\mathbb{E}$ operators to be interchanged. (18), used off-line during value iteration, consists of an independent collection of deterministic optimization problems, which may be run *in parallel* using off-the-shelf solvers. It is noted that the latter have been cornerstone of MPC technology. In this case, differentiable local averagers such as Kernel regression (Hastie et al. (2008)) may be employed. In this case, given the training set $\{x_i, \hat{J}(x_i)\}_{i=1}^N$, we have:

$$F\hat{J}(x_q) = \beta_0(x_q) + \frac{1}{\sum_{j=1} K\left(\frac{x_q - x_j}{\sigma}\right)} \cdot \sum_{i=1}^{N} K\left(\frac{x_q - x_i}{\sigma}\right)\hat{J}(x_i) \quad (20)$$

$\beta_0(x_q)$ is defined as in (15), where the Heaviside step function is replaced by a smooth approximator.

In addition to the off-line iteration step, the on-line calculation of the optimal input (in (19)) based on a pre-computed post-decision state cost-to-go function, is a deterministic optimization that does not involve an expectation operator, much like the one solved for MPC. Again, an off-the-shelf NLP solver may be employed. Another benefit, this time, compared to MPC, is that it involves only a single stage optimization as the cost-to-go function contains the precomputed optimal cost information for the rest of the horizon.

As with the pre-decision case, we can define a γ-contraction H^p so as to simplify (18). The aforementioned discussion on value-function approximation still holds for this case.

3.4 Adding new state samples: batch mode vs. continuous mode learning

In the standard ADP algorithm presented (that is, without Step (4)), we fix the set of sampled states, X_{sam}, in the beginning and do not introduce any more samples as the learning proceeds. One potential problem with this is that it may not contain sufficient samples in all the important regions of the state space.

Additional samples can be introduced as the learning proceeds in two different ways. First is to perform the simulation and the value iteration simultaneously, resulting in an X_{sam} that varies with simulation time. This approach is seen in the methods known as real-time dynamic programming (RTDP) (Barto et al. (1995)) and RTADP (Pratikakis et al. (2009)). In these approaches, one typically starts with an empty value table and introduces entries one by one as simulations proceed. Whenever a "new" state, a state that is not already recorded and does

not have a "sufficiently close" neighbor recorded in the value table, is visited during the simulation, it is entered into the value table and its cost-to-go value is assigned by evaluating the Bellman equation. The optimal action suggested by the current value table is implemented and next state is sampled according to the transition equation. If a state is revisited or there are sufficient neighbors close by, the value update for that particular state (or the sufficiently close neighbor) is performed without adding a new entry. This goes on until "new" entries are no longer added to the value table. Just how fast the convergence happens depends on the level of exploration, which will be discussed later.

The second way is to alternate between the modes of simulation and value iteration. A value iteration gives a converged cost-to-go function, which corresponds to a new policy. This new policy can be simulated to find a set of new state samples to be added to the current X_{sam}^{i} to yield X_{sam}^{i+1}. This continues until the simulation no longer yields different state samples.

In the case that an accurate simulation model is not available, one may have to replace the simulation with an actual on-line implementation. The continuous mode learning behaves much like adaptive control as the value table, and therefore the control policy, gets updated at every sample time. In addition, the performance during the initial phase of learning, when the value table has very few entries, may be highly unpredictable and poor. Hence, it may not be suitable in an industrial setting, whereas the other option in which the control policy remains fixed until the next off-line value iteration. Of course, given the typical constraints of industrial processes, one still has to exercise caution in implementing only a half-learned cost-to-go function on-line. The trade-off between exploration and robustness in this context is discussed next.

3.5 Exploration vs. robustness

The trade-off between exploration and robustness becomes one of the central issues when one chooses to expand X_{sam} as a part of the learning. In general, exploration gives new information to improve the eventual closed-loop performance (by expanding X_{sam}) but at the expense of slower convergence and decreased robustness. Exploration can be performed in two ways. First, one can add dither signals to the input to encourage more randomness in the state trajectories. Second, one can use an optimistic initialization of the cost-to-go value for previously unseen states, which will encourage the optimizer to choose actions leading the state trajectory to those states. If the learning is to be done directly in closed loop, the latter practice may be unacceptable for industrial practices as excursion to previously unseen states could jeopardize the safety and economics of the on-going operation. In other words, unlike in applications like robotics, "learning by mistake" is not a permissible practice for most industrial process control applications. One can in fact actively prevent such potentially harmful excursions through "pessimistic" valuation of unseen states. The previously discussed penalty approach in Section 3.2 is one way to achieve this. On the other hand, carefully chosen dither signals may be able to generate sufficiently new trajectories without imposing unacceptable risks or unduely slowing down the convergence.

3.6 Model-based vs. Model-less approach

For many industrial processes, sufficiently accurate models may not be available. In such a case, one can resort to empirical models derived from input-output data. In such as case, the state vector may simply be composed of the past input and output samples. The model can be learned separately from the ADP or it can be done as a part of it. In the latter, one learns instead of the cost-to-go function a function called Q function, which has the argument of state-input pair and assigns the cost-to-go to the pair. In other words, Q function already has the model embedded in it, which is learned together with the cost-to-go function. The two approaches are tried and compared in a recent paper by Lee and Lee (2005).

3.7 Integration with MPC

ADP can be integrated with model predictive control at several fronts. Some obvious ways include: (1)using MPC in the initial simulation to sample relevant states, (2)using the learned cost-to-go function in order to reduce the horizon size, and (3)use of the nonlinear programming solver of MPC in the post-decision-state formulation. Other methods may include the dual mode implementation, where MPC replaces the ADP controller whenever one encounters a state that is sufficiently new and the information in the learned value function cannot be trusted. Such states can be collected separately and added to X_{sam} in the next phase of value iteration.

4. EXAMPLES

Here, we demonstrate the proposed ADP algorithm on a variety of stochastic optimal control problems.

4.1 Example 1: Constrained linear stochastic system-double integrator problem

We consider the following constrained double integrator problem studied by (Batina (2004)) in the context of MPC for stochastic systems:

$$x_{t+1} = Ax_t + Bu_t + \Upsilon w_t$$

where matrix $A = [1\,0; 1\,1]$ [1], $B = [1; 0]$, $\Upsilon = [1; 0]$ and ω_t is zero-mean, white Gaussian noise with its second moment, $\mathbb{E}[\omega_t \omega_t'] = 0.2$, $\forall t$. The nominal stage-wise cost is $\tilde{\phi}(x_t, u_t) \triangleq 0.7||x_t||_2^2 + 0.33||u_t||_2^2$. The second dimension of the state vector is constrained, as is the input vector: $u_t \in [-0.5, 0.5]$, $x_2 \geq 0$.

The goal is to bring the system optimally from an arbitrary initial state ($[0; 14]$ in the following simulations) to the origin, whilst respecting the imposed constraints. We compare the performance of a Linear Model Predictive Controller (LMPC) (with prediction and control (p) horizon set to 15) against the proposed ADP approach based on the post-decision state variable. The post-decision state

[1] in Matlab notation

is defined as the quantity obtained after an action is taken but before the uncertainty is realized. That is, $x_t^p \triangleq Ax_t + Bu_t$. Since ω is an unbounded signal, we employ a soft-constraint approach for both LMPC (to avoid running into infeasibility issues) and the proposed ADP strategy. As is typically done, LMPC is implemented assuming ω remains at its nominal value of 0 over the prediction horizon. Namely, for LMPC, we solve at each time step, t:

$$\min \sum_{k=0}^{p} \tilde{\phi}(x_{t+k}, u_{t+k}) + 100||\epsilon_{t+k}||_2^2 \tag{21}$$

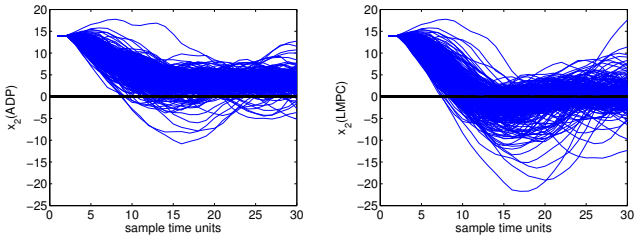
where $\epsilon_{t+k} \geq 0$ are non-negative auxiliary decision variables representing the least amount of slack required to make the LMPC problem feasible. That is, $[0\ 1]\, x_{t+k} + \epsilon_{t+k} \geq 0$. These inequalities are easily incorporated into the math program defined by (21). Also, the input vector is constrained to satisfy the aforementioned bounds of ± 0.5.

For the proposed ADP approach, we set the discount factor to a value close to unity, that is, $\gamma = 0.98$ and modify the stage wise cost to penalize deviations from the state constraints. Namely, $\phi(x_t, u_t) \triangleq \tilde{\phi}(x_t, u_t) + 100 \max(0, -[0\ 1]x_t)^2$. Hard constraints on u are imposed during the off-line value iteration process and on-line implementation of the ADP-based controller.

To construct X_{sam}, we used an LMPC controller (with horizon length 5) and conducted closed-loop experiments bringing the system from 40 different initial post-decision states to the origin. Note that the initial state used for on-line testing is excluded from these 40 initial states. Namely, we consider various combinations of the sets $\{-2, 0, -1, 1, 2\}$ and $\{-4, -2, 0, 2, 4, 6, 8, 10\}$ to create various values for the first and second dimension of the initial state respectively. Consequently, a total of 3587 training points, whose initial cost-to-go values were initialized by computing the cost for LMPC over a sufficiently long horizon, was obtained as a result of the initialization scheme. For the purpose of function approximation, we employed kernel regression with the bandwidth, σ, set to 0.16. To avoid over-extrapolation, we selected $A = 1220$, and $\rho = 0.2652$. Value-iteration converged within 50 iterations, where the relative error termination criterion is set to $\left|\left| \frac{\hat{J}^{\mu^*,p}_{[i+1]} - \hat{J}^{\mu^*,p}_{[i]}}{\hat{J}^{\mu^*,p}_{[i]}} \right|\right|_\infty \leq 0.1$.

Results from 500 stochastic realizations presented as follows. As can be seen from Table 1, the proposed ADP controller has an average [2] finite horizon score an order of magnitude lower than a deterministic approach typified by LMPC. In particular, LMPC suffers from excessively high variance in terms of closed-loop performance. A look at the time series plots of the second dimension of x for both methods (see Fig. 1) reveals that LMPC results in significant constraint violation. On the other hand, the majority of the realizations based on the ADP approach do not violate the lower bound constraint.

[2] based on sample averaging



(a) ADP algorithm. (b) LMPC.

Fig. 1. Double integrator example: x_2 vs. t for 500 realizations. Lower bound for x_2 is 0.

4.2 Example 2: Constrained nonlinear stochastic system - chemostat problem

Consider the governing equations of an archetypal chemostat.

$$\dot{x}_1 = x_1 \tfrac{\mu_{max} x_2}{\kappa + x_2} - x_1 u$$
$$\dot{x}_2 = u[x_{2,f} - x_2] - \tfrac{\mu_{max}}{Y} \tfrac{x_1 x_2}{\kappa + x_2}$$

where $x = [x_1; x_2] \in \mathbb{R}^2$ is the state vector composed of the instantaneous concentration of the product (x_1) and substrate (x_2) respectively. $0 \leq u \in \mathbb{R}$, the dilution rate, is the non-negative manipulated variable. $x_{2,f}$ refers to the instantaneous concentration of the substrate feed. The maximum specific growth rate μ_{max} is set to 1, the yield coefficient to 1 and κ to 0.02. For the following simulations, the sampling rate is set to 0.5.

For the purpose of simulation, we assume that the feed concentration ($x_{2,f}$) fluctuates around a mean value of 1, and is perturbed by zero-mean, white Gaussian noise (ω):

$$x_{2,f,t} = 1 + \omega_t,\ \mathbb{E}[\omega_t \omega_t'] = 10^{-3} \tag{22}$$

It is desirable to maximize the productivity of the product, $\mathcal{P}_t \triangleq x_{1,t} u_t$, whilst ensuring that the conversion of the substrate, $f_{x_2} \triangleq 1 - \frac{x_2}{x_{2,f}}$, does not go lower than a relatively high value of 95%. Such an economically motivated constraint is common in several key process industries, such as bioethanol production. There is a tradeoff between productivity and conversion. Productivity increases with dilution rate and then decreases as the system approaches washout. Conversion, on the other hand, is a decreasing function of space-velocity or equivalently the dilution rate. Maximum productivity ($\mathcal{P}^* = 0.7543$) occurs at a dilution rate that corresponds to conversion levels significantly below the required 95% threshold.

We compare the performance of Non-linear MPC (NMPC) against the proposed ADP strategy. Instead of full-fledged NMPC, we employ successive-linearization based MPC (slMPC), a computationally efficient alternative proposed by Lee and Ricker (1994). For this example, we have found the closed-loop performance of slMPC to be similar to that of NMPC. For slMPC, we employed a prediction and

Table 1. Example 1: comparing performance

Score	ADP	LMPC
$\mathbb{E}\left[\sum_{t=0}^{30} \phi(x_t, u_t)\right]$	1600	10000

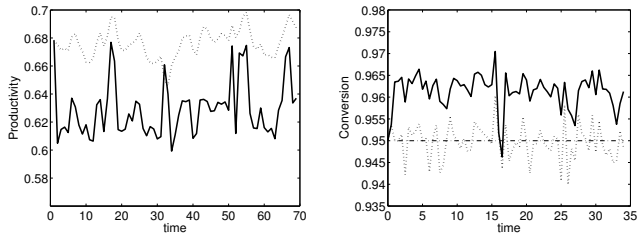
control horizon, p, of 10 sample units. The following math program is solved at each time instant:

$$\min \sum_{k=0}^{p} \left(||\mathcal{P}_{t+k} - \mathcal{P}^*||_2^2 + 100||\epsilon_{t+k}||_2^2 \right) \qquad (23)$$

where as in the previous example, $\epsilon \geq 0$, is a non-negative variable representing the least amount of slack required for conversion to be greater than 95%. That is, $\epsilon_{t+k} + f_{x_2,t+k} \geq 0.95$. The idea is to regulate the system at an equilibrium point that corresponds to the largest possible value of the dilution rate without exceeding the conversion bound so that productivity is maximized. The dynamics of the system are assumed to be governed by matrices obtained through linearization of the governing ordinary differential equations about the current state and past input vector. This results in a convex quadratic program.

For the proposed ADP approach, γ is set to 0.98 and the stage-wise cost defined as such: $\phi(x_t, u_t) \triangleq ||\mathcal{P}_t - \mathcal{P}^*||_2^2 + 100 \max(0, 0.95 - f_{x_2,t})^2$. To determine X_{sam}, we used an slMPC controller (with horizon length of 10 time units) and conducted closed-loop experiments regulating the system at an initial state corresponding to a conversion of 0.95. A total of 300 training points was obtained from the initialization scheme. We used kernel regression for function approximation with the bandwidth, σ, set to 0.15, A to 1.93 and ρ to 0.087 in order to prevent over-extrapolation. Value iteration terminated within 50 iterations with a relative error tolerance of 0.1.

Results from a typical realization are depicted in Fig. 2. It is apparent that the ADP-based approach, compared to slMPC, results in minimal constraint violation at the expense of slightly lower productivity. It is noted that the steady-state productivity corresponding to 95% conversion is 0.68.



(a) Productivity vs time. (b) Conversion vs time.

Fig. 2. Example 2. Closed-loop performance of a typical realization. ADP: solid line (-); slMPC: dotted line(..); lower bound on conversion: dash-dot (-.)

4.3 Other examples in the literature

There are a number of other applications to process process control problems in the published literature. Interested readers may look at the following references for applications to more complex examples. These include: integrated reactor-separator system control (Tosukhowong and Lee (2009)), dual adaptive control (Lee and Lee (2009)), fed-batch reactor control (Peroni et al. (2005)), and microbial reactor (Kaisare et al. (2003)).

5. CONCLUSIONS

We have examined the potentials of ADP for process control and found that it can complement MPC to reduce the on-line computational load and also address stochastic system uncertainties. ADP offers a number of design options and one must think carefully through them to choose the right options for a given application. We have argued that, for process control problems, post-decision-state formulation offers the ability to use deterministic math programming solvers to be utilized, both off-line and on-line and therefore may be more convenient than the more conventional pre-decision-state formulation. In addition, the use of function approximators with nonexpansion properties offer stable learning. Robustness against over-extrapolation can be achieved through the use of a tailor-made penalty function. Finally, to achieve performance close to optimal ones, we recommend alternation between the value function update and simulation (or on-line implementation) to increase the sample set as the learning proceeds.

Though not discussed in this paper, there are a number of other application areas within process industries where ADP can prove to be a valuable tool, including resource allocation and inventory management (Pratikakis et al. (2008, 2009); Choi et al. (2004, 2006)), design and planning under uncertainty (Cheng et al. (2003)), scheduling of multiple controllers (Lee and Lee (2008)), and equipment / product inspection (Agrawal (2009)). Raised awareness of the ADP technique within the process systems engineering research community will undoubtedly bring forth additional applications that can benefit from it.

REFERENCES

Agrawal, R. (2009). *Planning and scheduling problems in manufacturing systems with high degree of resource degradation*. Ph.D. thesis, Georgia Institute of Technology.

Barto, A., Bradtke, S., and Singh, S. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1), 81–138.

Batina, I. (2004). *Model predictive control for stochastic systems by randomized algorithms*. Ph.D. thesis, Technische Universiteit Eindhoven.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, New Jersey.

Bertsekas, D.P. (2005). *Dynamic Programming and Optimal Control, Vol. I*. Athena-Scientific, Belmont, MA, 3 edition.

Bertsekas, D. and Tsitsiklis, J. (1996). *Neuro-dynamic programming*. Athena Scientific, 1 edition.

Boyan, J.A. and Moore, A.W. (1995). Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems 7*, 369–376. MIT Press.

Bradtke, S. (1993). Reinforcement learning applied to linear quadratic regulation. In *Adavances in neural information processing systems*, 295–302.

Cassandra, A., Kaelbling, L.P., and Littman, M.L. (1994). Acting optimally in partially observable stochastic domains. In *Twelfth National Conference on Artificial Intelligence*. Seattle, WA.

Cheng, L., Subrahmaniam, E., and Westerberg, A. (2003). Design and planning under uncertainty: issues on problem formulation and solution. *Computers & Chemical Engineering*, 27, 781–801.

Choi, J., Realff, M.J., and Lee, J.H. (2004). Dynamic programming in a heuristically confined state space: a stochastic resource constrained project scheduling application. *Computers & Chemical Engineering*, 8, 1039–1058.

Choi, J., Realff, M.J., and Lee, J.H. (2006). Dynamic programming in a heuristically restricted state space: stochastic supply chain management application. *AIChE Journal*, 52(7), 2473–2485.

De Farias, D.P. and Van Roy, B. (2003). The linear programming approach to approximate dynamic progrmaming. *Operations research*, 51(6), 850–865.

Gordon, G.J. (1995). Stable function approximation in dynamic programming. Technical report, School of Computer Science, Carnegie Mellon University.

Hastie, T., Tibshirani, R., and Friedman, J. (2008). *The Elements of Statisical Learning: Data Mining, Inference and Prediction*. Springer-Verlag, 2 edition.

Kaisare, N., Lee, J.M., and Lee, J.H. (2003). Simulation based strategy for nonlinear optimal control: application to a microbial cell reactor. *International journal of nonlinear and robust control*, 13, 347–363.

Konidaris, G. and Osentoski, S. (2008). Value function approximation in reinforcement learning using the fourier basis. Technical report, University of Massachusetts Amherst.

Laird, C.D. and Biegler, L.T. (2008). Large-scale nonlinear programming for multi-scenario optimization. In H.G. Bock, E. Kostina, H.X. Phu, and R. Rannacher (eds.), *Modeling, Simulation and Optimization of Complex Processes*, 323–336. Springer Berlin Heidelberg.

Lee, J.H. and Lee, J.M. (2006). Approximate dynamic programming based approach to process control and scheduling. *Computers & Chemical Engineering*, 30(10-12), 1603–1618.

Lee, J. and Ricker, N. (1994). Extended kalman filter based nonlinear model predictive control. *Industrial & Engineering Chemistry Research*, 33(6), 1530–1541.

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

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

Lee, J.M. and Lee, J.H. (2008). Value function-based approach to the scheduling of multiple controllers. *Journal of process control*, 18, 533–542.

Lee, J.M. and Lee, J.H. (2009). An approximate dynamic programming approach based approach to dual adaptive control. *Journal of process control*, 19, 859–864.

Ma, J. and Powell, W. (2009). A convergent recursive least squares policy iteration algorithm for multi-dimensional markov decision process with continuous state and action spaces. In *IEEE Conference on Approximate Dynamic Programming and Reinforcement Learning (part of IEEE Symposium on Computational Intelligence)*. Nashville, TN.

Ormoneit, D. and Sen, S. (2002). Kernel-based reinforcement learning. *Machine Learning Journal*, 49, 161–178.

de la Penad, D.M., Bemporad, A., and Alamo, T. (2005). Stochastic programming applied to model predictive control. In *44th IEEE Conference on Decision and Control, and the European Control Conference*, 1361–1366. Seville, Spain.

Peroni, C., Kaisare, N., and Lee, J.H. (2005). Optimal control of a fed batch bioreactor using simulation-based approximate dynamic programming. *IEEE Transactions in Control Systems Technology*, 13, 786–790.

Powell, W.B. (2007). *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Wiley Series in Probability and Statistics. Wiley-Interscience.

Pratikakis, N., Realff, M.J., and Lee, J.H. (2008). A controlled exploration of the state space via an off-line adp approach: applications on stochastic shortest path and manufacturing systems. *Computers & Chemical Engineering*, in press.

Pratikakis, N., Realff, M.J., and Lee, J.H. (2009). Strategic capacity decisions in manufacturing using real-time adaptive dynamic programming. *Naval Research Logistics*, to be published.

Putterman, M.L. (1994). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley and Sons, New York.

Roy, B.V., Bertsekas, D.P., Lee, Y., and Tsitsiklis, J. (1997). A neuro-dynamic programming approach to retailer inventory management. In *IEEE Conference on Decision and Control*, volume 4, 4052–4057.

Sabes, P. (1993). Approximating q-values with basis function representations. In *Fourth Connectionist Models Summer School*. Hillsdale, NJ.

Scokaert, P. and Mayne, D. (1998). Min-max feedback model predictive control for constrained linear systems. *IEEE Transactions of Automatic Control*, 43(8), 1136–1142.

Smart, W. and Kaelbling, L. (2000). Practical reinforcement learning in continuous spaces. In *17th international conference on machine learning*, 903–910.

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

Tesauro, G. (1992). Practical issues in tempora-difference learning. *Machine Learning Journal*, 8, 257–277.

Thrun, S. and Schwartz, A. (1993). Issues in using function approximation for reinforcement learning. In *Fourth Connectionist Models Summer School*. Hillsdale, NJ.

Tosukhowong, T. and Lee, J.H. (2009). Approximate dynamic programming based optimal control applied to an integrated plant with a reactor and a distillation column with recycle. *AIChE Journal*, 55(4), 919–930.

Tsitsiklis, J. and Roy, B.V. (1996). Feature-based methods for large scale dynamic programming. *Machine Learning Journal*, 22(1), 59–94.