Approximate Dynamic Programming Strategies and Their Applicability for Process Control: A Review and Future Directions

Jong Min Lee and Jay H. Lee*

Abstract: This paper reviews *dynamic programming* (DP), surveys approximate solution methods for it, and considers their applicability to process control problems. Reinforcement Learning (RL) and Neuro-Dynamic Programming (NDP), which can be viewed as approximate DP techniques, are already established techniques for solving difficult multi-stage decision problems in the fields of operations research, computer science, and robotics. Owing to the significant disparity of problem formulations and objective, however, the algorithms and techniques available from these fields are not directly applicable to process control problems, and reformulations based on accurate understanding of these techniques are needed. We categorize the currently available approximate solution techniques for dynamic programming and identify those most suitable for process control problems. Several open issues are also identified and discussed.

Keywords: Approximate dynamic programming, reinforcement learning, neuro-dynamic programming, optimal control, function approximation.

1. INTRODUCTON

Dynamic programming (DP) offers a unified approach to solving multi-stage optimal control problems [12,13]. Despite its generality, DP has largely been disregarded by the process control community due to its unwieldy computational complexity referred to as 'curse-of-dimensionality.' As a result, the community has studied DP only at a conceptual level [55], and there have been few applications.

In contrast, the artificial intelligence (AI) and machine learning (ML) fields over the past two decades have made significant strides towards using DP for practical problems, as evidenced by several recent review papers [34,95] and textbooks [18,86]. Though a myriad of approximate solution strategies have been suggested in the context of DP, they are mostly tailored to suit the characteristics of applications in operations research (OR), computer science, and robotics. Since the characteristics and requirements of these applications differ considerably from those of process control problems, these approximate methods should be understood and interpreted carefully from the viewpoint of process control before they can be considered for real process control problems.

The objective of this paper is to give an overview of the popular approximation techniques for solving DP developed from the AI and ML fields and to summarize the issues that must be resolved before these techniques can be transferred to the field of process control.

The rest of the paper is organized as follows. Section 2 motivates the use of DP as an alternative to model predictive control (MPC), which is the current state-of-the-art process control technique. We also give the standard DP formulation and the conventional solution approaches. Section 3 discusses popular approximate solution strategies, available mainly from the reinforcement learning (RL) field. Important applications of the strategies are categorized and reviewed in Section 4. Section 5 brings forth some outstanding issues to be resolved for the use of these techniques in process control problems. Section 6 summarizes the paper and points to some future directions.

2. DYNAMIC PROGRAMMING

2.1. Alternative to Model Predictive Control (MPC)?

Process control problems are characterized by complex multivariable dynamics, constraints, and competing sets of objectives. Because of the MPC's ability to handle these issues systematically, the technique has been widely adopted by process industries.

Manuscript received May 29, 2004; accepted July 14, 2004. Recommended by Editor Keum-Shik Hong under the direction of Editor-in-Chief Myung Jin Chung.

Jong Min Lee and Jay H. Lee are with the School of Chemical and Biomolecular Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0100, USA (e-mails: {Jongmin.Lee, Jay.Lee}@chbe.gatech.edu).
* Corresponding author.

MPC is based on periodic execution of state estimation followed by solution of an *open-loop* optimal control problem formulated on a finite moving prediction horizon [55,66]. Notwithstanding its impressive track record within the process industry, a close scrutiny points to some inherent limitation in its versatility and performance. Besides the obvious limitation imposed by the need for heavy on-line computation, the standard formulations are based on solving an *open-loop* optimal control problem at each sample time, which is a poor approximation of the *closed-loop* control MPC actually performs in an uncertain environment. This mismatch between the mathematical formulation and the reality can translate into substantial performance losses.

DP allows us to derive an optimal *feedback* control policy *off-line* [12,65,13], and hence has the potentials to be developed into a more versatile process control technique without the shortcomings of MPC. The original development of DP by Bellman dates back to the late 50s. Though used to derive celebrated results for simple control problems such as linear quadratic (Gaussian) optimal control problem, DP has largely been considered to be impractical by the process control community because of the so called 'curse-of-dimensionality,' which refers to the exponential growth in the computation with respect to the state dimension. The computational and storage requirements for almost all problems of practical interest remain unwieldy even with today's computing hardware.

2.2. Formulation of DP

Consider an optimal control problem with a predefined stage-wise cost and state transition equation. The sets of all possible states and actions are represented by X and U, respectively. In general, $X \subset R^{n_x}$ and $U \subset R^{n_u}$ for process control problems, where n_x and n_u are the number of state and action variables, respectively.

Let us denote the system state at the k^{th} time step by $x(k) \in X$, the control action by $u(k) \in U$, and some random disturbance by $\omega(k) \in R^{n_\omega}$. The successor state $x(k+1)$ is defined by the transition equation of

$$x(k+1) = f\left(x(k), u(k), \omega(k)\right). \tag{1}$$

Note that (1) represents a Markov decision process (MDP), meaning the next state $x(k+1)$ depends only on the current state and input not on the states and actions of past times. Throughout the paper, a model in the form of (1) will be assumed. The model form is quite general in that, if the next state did indeed depend on past states and actions, a new state vector can be defined by including those past variables. Whereas (1) is the typical model form used for process control problems, most OR problems have a model described by a transition probability matrix, which describes how the probability distribution (over a finite set of discrete states) evolves from one time step to the next. We also assume that a bounded single-stage cost, $\varphi\left(x(k), u(k)\right)$, is incurred immediately or within some fixed sample time interval after a control action $u(k)$ is implemented. We restrict the formulation to the case where state information is available either from direct measurements or from an estimator. A control policy μ is defined to map a state to a control action, i.e., $u = \mu(x)$. We also limit our investigation to stationary (time-invariant) policies and infinite horizon objective functions. DP for finite horizon problems are also straightforward to formulate [13]. For any given control policy μ, the corresponding infinite horizon cost function is defined as

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

where E_{μ} is the conditional expectation under the policy $u = \mu(x)$, and $\alpha \in [0,1)$ is a discount factor that handles the tradeoff between the immediate and delayed costs. $J^{\mu}(x)$ represents the expected discounted total cost starting with the state x. Note that J^{μ} is a function of x, which will be referred to as the 'cost-to-go' function hereafter. Also define

$$J^*(x) = \inf_{\mu} J^{\mu}(x) \qquad \forall x \in X. \tag{3}$$

A policy μ^* is α-optimal, if

$$J^{\mu^*}(x) = J^*(x) \qquad \forall x \in X. \tag{4}$$

The closed-loop (state feedback) optimal control problem is formulated as a dynamic program yielding the following function equation:

$$J^*(x(k)) = \min_{u(k) \in U} E\left[\varphi(x(k), u(k)) + \alpha J^*(x(k+1))\right], \quad \forall x \in X. \tag{5}$$

(5) is called *Bellman equation* and its solution defines the optimal cost-to-go function for the entire state space. The objective is then to solve the Bellman equation to obtain the optimal cost-to-go function,

which can be used to define the optimal control policy as follows:

$$\mu^*(x(k)) = \arg\min_{u(k)\in U} E\left[\varphi(x(k),u(k)) + \alpha J^*(x(k+1))\right], \quad \forall x \in X. \quad (6)$$

2.3. Value/Policy iteration

In this section, we review two conventional approaches for solving DP, value iteration and policy iteration. They form the basis for the various approximate solution methods introduced later.

- Value iteration

  In value iteration, one starts with an initial guess for the cost-to-go for each state and iterates on the Bellman equation until convergence. This is equivalent to calculating the cost-to-go value for each state by assuming an action that minimizes the sum of the current stage cost and the 'cost-to-go' for the next state according to the current estimate. Hence, each update assumes that the calculated action is optimal, which may not be true given that the cost-to-go estimate is inexact, especially in the early phase of iteration. The algorithm involves the following steps.

  1. Initialize $J^0(x)$ for all $x \in X$.
  2. For each state x

$$J^{i+1}(x) = \min_u E\left[\varphi(x,u) + \alpha J^i(\hat{x})\right] \quad (7)$$

  where $\hat{x} = f(x,u,\omega) \in X$, and i is the iteration index.

  3. Perform the above iteration (step 2) until $J(x)$ converges.

  The update rule of (7) is called *full backup* because the cost-to-go values of the entire state space are updated in every round of update.

- Policy iteration

  Policy iteration is a two-step approach composed of *policy evaluation* and *policy improvement*. Rather than solve for a cost-to-go function directly and then derive an optimal policy from it, the policy iteration method starts with a specific policy and the policy evaluation step computes the cost-to-go values under that policy. Then the policy improvement step tries to build an improved policy based on the cost-to-go function of the previous policy. The policy evaluation and improvement steps are repeated until the policy no longer changes. Hence, this method iterates on policy rather than cost-to-go function.

  The policy evaluation step iterates on the cost-to-go values but with the actions dictated by the given policy. Each evaluation step can be summarized as follows:

  1. Given a policy μ, initialize $J_\mu(x)$ for all $x \in X$.
  2. For each state x

$$J_\mu^{\ell+1}(x) = E\left[\varphi(x,\mu(x)) + \alpha J_\mu^\ell(\hat{x})\right]. \quad (8)$$

  3. The above iteration (step 2) continues until $J_\mu(x)$ converges.

  The overall policy iteration algorithm is given as follows:

  1. Given an initial control policy μ^0, set $i = 0$.
  2. Perform the policy evaluation step to evaluate the cost-to-go function for the current policy μ^i.
  3. The improved policy is represented by

$$\mu^{i+1}(x) = \arg\min_u E\left[\varphi(x,u) + \alpha J_{\mu^i}(\hat{x})\right]. \quad (9)$$

  Calculate the action given by the improved policy for each state.

  4. Iterate steps 2 and 3 until $\mu(x)$ converges.

For systems with a finite number of states, both the value iteration and policy iteration algorithms converge to an optimal policy [12,32,13]. Whereas policy iteration requires complete policy evaluation between steps of policy improvement, each evaluation often converges in just few iterations because the cost-to-go function typically changes very little when the policy is only slightly improved. At the same time, policy iteration generally requires significantly fewer policy improvement steps than value iteration because each policy improvement is based on accurate cost-to-go information [65].

One difficulty associated with value or policy iteration is that the update is performed after one "sweep" of an entire state set, making it prohibitively expensive for most problems. To avoid this difficulty, *asynchronous* iteration algorithms have been proposed [17,16]. These algorithms do not back up the values of states in a strict order but use whatever updated values available. The values of some states may be backed up several times while the values of others are backed up once. However, obtaining optimal cost-to-go values requires infinite number of updates in general.

2.4. Linear programming based approach

Another approach to solving DP is to use a linear programming (LP) formulation. The Bellman equation can be characterized by a set of linear constraints on the cost-to-go function. The optimal cost-to-go function, which is the fixed point solution of the Bellman equation, can also be obtained by solving a LP [48,26,30,19]. Let us define a DP operator T to simplify (5) as follows:

$$J = TJ. \tag{10}$$

Then J^* is the unique solution of the above equation. Solving this Bellman equation is equivalent to solving the following LP:

$$\max \quad c^T J, \tag{11}$$

$$\text{s.t.} \quad TJ \geq J, \tag{12}$$

where c is a positive weight vector. Any feasible J must satisfy $J \leq J^*$, and therefore for any strictly positive c, J^* is the unique solution to the LP of (11). Note that (12) is a set of constraints

$$E\left[\varphi(x,u) + \alpha J(\hat{x})\right] \geq J(x), \quad \forall u \in U \tag{13}$$

leaving us with the same 'curse-of-dimensionality'; the number of constraints grows with the number of states and the number of possible actions.

The LP approach is the only known algorithm that can solve DP in polynomial time, and recent years have seen substantial advances in algorithms for solving large-size linear programs. However, theoretically efficient algorithms have still been shown to be ineffective or even infeasible for practically-sized problems [34,86].

3. APPROXIMATE METHODS FOR SOLVING DP

Whereas the process control community concluded DP to be impractical early on, researchers in the fields of machine learning and artificial intelligence began to explore the possibility of applying the theories of psychology and animal learning to solving DP in an approximate manner in the 1980s [85,80]. The research areas related to the general concept of programming agents by "reward and punishment without specifying how the task is achieved" have collectively been known as 'reinforcement learning (RL)' [86,34]. It has spawned a plethora of techniques to teach an agent to learn cost or utility of taking actions given a state of the system. The connection between these techniques and the classical dynamic programming was elucidated by Bertsekas and Tsitsiklis [18,95], who coined the term *Neuro-Dynamic Programming* (NDP) because of the popular use of artificial neural networks (ANNs) as the function approximator.

In the rest of this section, we first discuss the representation of state space, and then review different approximate DP algorithms, which are categorized into model-based and model-free methods. The most striking feature shared by all the approximate DP techniques is the synergetic use of *simulations* (or interactive experiments) and *function approximation*. Instead of trying to build the cost-to-go function for an entire state space, they use sampled trajectories to identify parts of the state space relevant to optimal or "good" control where they want to build a solution and also obtain an initial estimate for the cost-to-go values.

3.1. State space representation

Typical MDPs have either a very large number of discrete states and actions or continuous state and action spaces. Computational obstacles arise from the large number of possible state/action vectors and the number of possible outcomes of the random variables. The 'curse-of-dimensionality' renders the conventional DP solution approach through exhaustive search infeasible. Hence, in addition to developing better learning algorithms, substantial efforts have been devoted to alleviating the curse-of-dimensionality through more compact state space representations. For example, state space quantization/discretization methods have been used popularly in the context of DP [9] and gradient descent technique [79]. The discretization/ quantization methods have been commonly accepted because the standard RL/NDP algorithms were originally designed for systems with discrete states. The discretization method should be chosen carefully, however, because incorrect discretization could severely limit the performance of a learned control policy, for example, by omitting important regions of the state space and/or by affecting the original Markov property [56].

More sophisticated discretization methods have been developed based on adaptive resolutions such as the multi-level method [67], clustering-based method [38], triangularization method [56,57], state aggregation [15], and divide-and-conquer method (Parti-Game algorithm) [52]. The parti-game algorithm, which is one of the most popular discretization strategies, attempts to search for a path from an initial state to a goal state (or region) in a multi-dimensional state space based on a coarse discretization. When the search fails, the resolution is iteratively increased for the regions of the state space where the path planner is unsuccessful. Though some adaptive discretization methods can result in a better

policy compared to the fixed versions [6], they can potentially suffer from their own 'curse-of-dimensionality' and become less reliable when the estimation of cost-to-go is necessary for the states lying in a smaller envelope than that of converged partitions.

Function approximation methods have also been employed to represent the relationship between cost-to-go and system state in a continuous manner either by based on parametric structures (e.g. ANNs) or 'store-and-search' based nonparametric methods (e.g. nearest neighbor). The function approximation based approaches are more general because they are applicable to both finite and infinite number of states without modification of a given problem. The current status and the issues of incorporating function approximators into approximate DP strategies will be discussed separately in section 5.

3.2. Model-based methods

If there exists a model that describes a concerned system, the main question becomes how to solve the Bellman equation efficiently. Given an exact model, a conceptually straightforward approach is to use the value or policy iteration algorithm. Since it is not feasible to do this for all states, one plausible approach is to use a set of sampled data from closed-loop simulations (under known suboptimal policies) to reduce the number of states for which the Bellman equation is solved. Since the successor states during each iteration may be different from the ones in the simulations, function approximation is employed to estimate the cost-to-go values for those states not visited during the simulation runs [35,43]. A family of methods in which a model built from data is used to derive a control policy as if it were an exact representation of the system is called 'certainty-equivalence' approach, which is similar to the concept for process identification/control involving *learning phase* and *acting phase* [39]. We note that random exploration for gathering data to build such a model is much less efficient than policy-interlaced exploration [100,36].

Independently from the researchers working on direct DP solution methods, Werbos proposed a family of adaptive critic designs (or actor-critic methods (AC) as named later in [11]) in the late 1970s [98]. He extended the work and collectively called the approach *Heuristic Dynamic Programming* [99]. The purpose of the adaptive critic design is to learn optimal control laws by successively adapting two ANNs, namely, an action network and a critic network. These two ANNs indirectly approximate the Bellman equation. The action network calculates control actions using the performance index from the critic network. The critic network learns to approximate the cost-to-go function and uses the output of the action

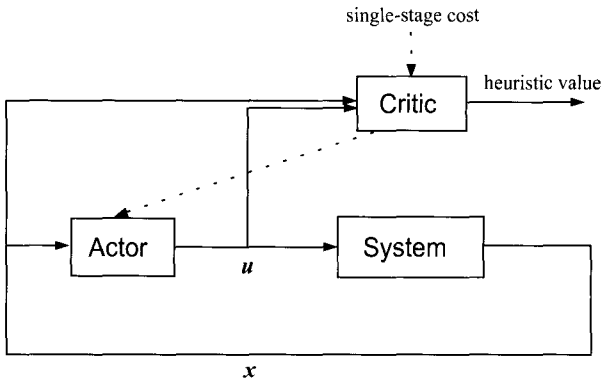


Fig. 1. The actor-critic architecture.

network as one of its inputs, either directly or indirectly. This structure has been used as a "policy learner" in conjunction with many RL schemes in addition to being a popular structure for neuro-fuzzy-controllers [64]. The AC algorithms are less suited to cases where the data change frequently since the training of the networks is challenging and time-consuming. We note that the AC framework is not limited to the model-based learning scheme, and it has also been used as a framework for model-free learning. The general structure of AC is shown in Fig. 1.

RL literature has considered the model-based learning as an alternative way to use gathered data efficiently during interactive learning with an environment, compared to a class of model-free learning schemes that will be introduced in the next section. They have been interested in 'exploration through trial-and-error' to increase the search space, for example, in a robot-juggling problem [74]. Hence, most model-based approaches from the RL literature have been designed to learn an explicit model of a system simultaneously with a cost-to-go function and a policy [82,83,54,63,10]. The general algorithms iteratively 1) update the learned model, 2) calculate control actions that optimize the given cost-to-go function with the current learned model, 3) update the corresponding cost-to-go function as in value iteration, and 4) execute the control policy and gather more data. Representative algorithms in this class are Dyna and RTDP (real-time dynamic programming) [82,10]. These model-based interactive learning techniques have the advantage that they can usually find good control actions with fewer experiments since they can exploit the existing samples better by using the model [27].

3.3. Model-free methods

RL/NDP and other related research work have mainly been concerned with the question of how to obtain an optimal policy when a model is not available. This is mainly because the state transition rule of their concerned problems is described by a

probability transition matrix, which is difficult to identify empirically. Many trial-and-errors, however, allow one to find the optimal policy eventually. These "on-line planning" methods have an agent interact with its environment directly to gather information (state and action vs. cost-to-go) from on-line experiments or in simulations. In this section, three important model-free learning frameworks are introduced – Temporal Difference (TD) learning, Q-learning, and SARSA, all of which learn the cost-to-go functions *incrementally* based on experiences with the environment.

3.3.1 Temporal difference learning

TD learning is a passive learner in that one calculates the cost-to-go values by operating an agent under a *fixed* policy. For example, we watch a robot wander around using its current policy μ to see what cost it incurs and which states it explores. This was suggested by Sutton and is known as the TD(0) algorithm [81]. The general algorithm is as follows:

1. Initialize $J(x)$.
2. Given a current policy ($\mu^i(x)$), let an agent interact with its environment, for example, let an agent (e.g. robot, controller, etc.) perform some relevant tasks.
3. Watch the agent's actions given from $\mu^i(x)$, obtain a cost $\varphi(x,\mu^i(x))$, and its successor state $\hat{x}$.
4. Update the cost-to-go using

$$J(x) \leftarrow (1-\gamma)J(x) + \gamma\left\{\varphi(x,\mu^i(x)) + \alpha J(\hat{x})\right\} \quad (14)$$

or equivalently,

$$J(x) \leftarrow J(x) + \gamma\left\{\varphi(x,\mu^i(x)) + \alpha J(\hat{x}) - J(x)\right\} \quad (15)$$

where γ is a learning rate from 0 to 1. The higher the γ, the more we emphasize our new estimates and forget the old estimates.

5. Set $x \leftarrow \hat{x}$ and continue experiment.
6. If one sweep is completed, return to step 2 with $i \leftarrow i+1$, and continue the procedure until convergence.

Whereas TD(0) update is based on the "current" difference 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.

Suppose we generated a sample trajectory, $\{x(0), x(1), \cdots, x(t), \cdots\}$. The temporal difference term, $d(t)$, at time t is given by

$$d(t) = \varphi(x(t),\mu(x(t))) + \alpha J(x(t+1)) - J(x(t)) \,. \quad (16)$$

Then the policy evaluation step for a stochastic system is approximated by

$$J(x(t)) \leftarrow J(x(t)) + \gamma \sum_{m=t}^{\infty} \lambda^{m-t} d(m), \quad (17)$$

where $0 \le \lambda < 1$ is a decay parameter. Within this scheme, a single trajectory can include a state, say x, multiple times, for example, at times $t_1, t_2, \cdots, t_M$. In such a case, 'every-visit' rule updates the cost-to-go whenever the state is visited in the trajectory according to

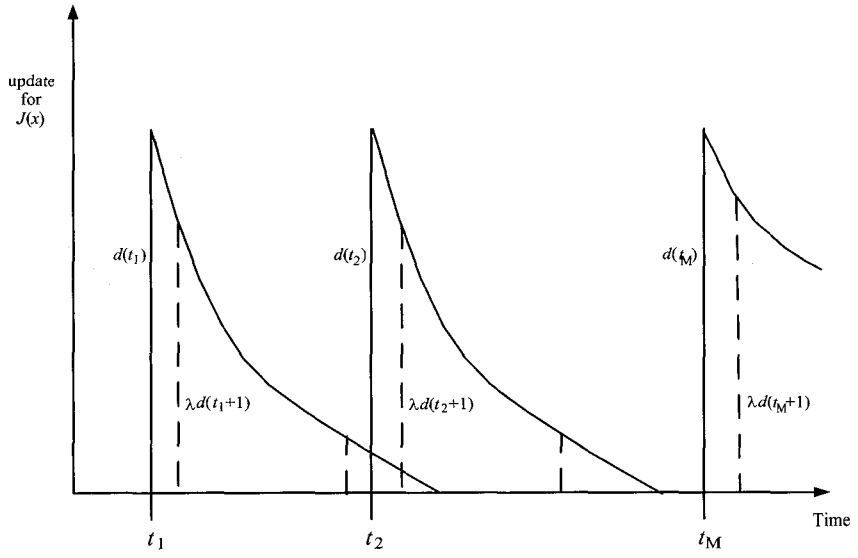


Fig. 2. Cumulative addition of temporal difference terms in the every-visit method.

$$J(x(t)) \leftarrow J(x(t)) + \gamma \sum_{j=1}^{M} \sum_{m=t_j}^{\infty} \lambda^{m-t_j} d(m). \quad (18)$$

Graphical representation of the addition of update terms is shown in Fig. 2.

A corresponding *on-line* update rule for the every-visit method is given by

$$J(x(t)) \leftarrow J(x(t)) + \gamma\{\varphi(x(t), \mu^i(x(t))) + \alpha J(x(t+1)) - J(x(t))\} e_t(x(t)), \quad (19)$$

where t is the time index in a single sample trajectory, and $e_t(x)$ is 'eligibility,' with which each state is updated. Note that the temporal difference term, $\{\cdot\}$ of (19), appears only if x has already been visited in a previous time of the trajectory. Hence, all eligibilities start out with zeros and are updated at each time t as follows:

$$e_t(x) = \begin{cases} \alpha\lambda e_{t-1}(x) & \text{if} \quad x \neq x(t), \\ \alpha\lambda e_{t-1}(x) + 1 & \text{if} \quad x = x(t), \end{cases} \quad (20)$$

where α is the discount factor for the cost-to-go. The eligibility thereby puts more emphasis on the temporal difference term in recent past. Singh and Sutton [77] proposed an alternative version of the eligibility assignment algorithm, where visited states in the most recent sample run always get an eligibility of unity rather than an increment of 1, which they called the 'first-visit' method.

Since the TD learning is based on a fixed policy, it can be combined with an AC-type policy-learner. The convergence properties of the AC-related algorithms were explored [101]. The convergence property of TD(λ) learning was also studied by several researchers [24,62,93].

3.3.2 Q-learning

Q-learning [97,96] is an active learner in that one modifies the 'greedy' policy as the agent learns. One can also tweak the policy to try different control actions from the calculated policy even when the agent is interacting with a real environment. For example, injection of random signals into actions is often carried out for exploration of the state space. Optimal Q-value is defined as the cost-to-go value of implementing a specific action u at state x, and then following the optimal policy from the next time step on. Hence, the optimal Q-function satisfies the following equation:

$$Q^*(x,u) = E\left[\varphi(x,u) + \alpha \min_{\hat{u}} Q^*(\hat{x},\hat{u})\right] \quad (21)$$

$$= E\left[\varphi(x,u) + \alpha J^*(\hat{x})\right]. \quad (22)$$

This also gives a recursive relation for the Q-function, similarly as does the Bellman equation for the 'J-function.' Once the optimal Q-function $Q^*(x,u)$ is known, the optimal policy $\mu^*(x)$ can easily be obtained by

$$\mu^*(x) = \arg\min_u Q^*(x,u). \quad (23)$$

The on-line incremental learning of the Q-function is similar to the TD-learning:

1. Initialize $Q(x,u)$.
2. Let an agent interact with its environment by solving (23) using the current approximation of Q instead of Q^*. If there are multiple actions giving a same level of performance, select an action randomly.
3. Update the Q values by

$$Q(x,u) \leftarrow (1-\gamma)Q(x,u) + \gamma\left\{\varphi(x,u) + \alpha \min_{\hat{u}} Q(\hat{x},\hat{u})\right\}. \quad (24)$$

4. Set $x \leftarrow \hat{x}$ and continue the experiment.
5. Once a loop is complete, repeat from step 2 until convergence.

If one performs the experiment infinite times, the estimates of Q-function converge to $Q^*(x,u)$ with proper decaying of the learning rate γ [97,33]. Greedy actions may confine the exploration space, especially in the early phase of learning, leading to a failure in finding the optimal Q-function. To explore the state space thoroughly, random actions should be carried out on purpose. As the solution gets improved, the greedy actions are implemented. This randomization of control actions are similar to the simulated annealing technique used for global optimization.

3.3.3 SARSA

SARSA [68] also tries to learn the state-action value function (Q-function). It differs from Q-learning with respect to the incremental update rule. SARSA does not assume that the optimal policy is imposed after one time step. Instead of finding a greedy action, it assumes a fixed policy as does the TD learning. The update rule then becomes

$$Q(x,u) \leftarrow Q(x,u) + \gamma\left\{\varphi(x,u) + \alpha Q(\hat{x}, \mu^i(\hat{x})) - Q(x,u)\right\}. \quad (25)$$

A policy learning component like the AC scheme can also be combined with this strategy.

4. APPLICATIONS OF APPROXIMATE DP METHODS

In this section, we briefly review some of the important applications of the approximate DP methods, mainly RL and NDP methods. Important OR applications are reviewed in [18,86]. We classify the previous work by application areas.

4.1. Operations research

The application area that benefited the most from the RL/NDP theory is game playing. Samuel's checker player was one of the first applications of DP in this field and used linear function approximation [70,71]. One of the most notable successes is Tesauro's backgammon player, TD-Gammon [88-90]. It is based on TD methods with approximately 10^{20} states. To handle the large number of state variables, ANN with a simple feed forward structure was employed to approximate the cost-to-go function, which maps a board position to the probability of winning the game from the position. Two versions of learning were performed for training the TD-Gammon. The first one used a very basic encoding of the state of the board. The advanced version improved the performance significantly by employing some additional human-designed features to describe the state of the board. The learning was done in an evolutionary manner over several months – playing against itself using greedy actions without exploration. TD-Gammon successfully learned to play competitively with world-champion-level human players. By providing large amounts of data frequently and realizing the state transitions in a sufficiently stochastic manner, TD-Gammon could learn a satisfactory policy without any explicit exploration scheme. No comparable successes to that of TD-Gammon has been reported in other games yet, and there are still open questions regarding how to design experiments and policy update in general [75,91].

Another noteworthy application was in the problem of elevator dispatching. Crites and Barto [22,23] used Q-learning for a complex simulated elevator scheduling task. The problem is to schedule four elevators operating in a building with ten floors. The objective is to minimize the discounted average waiting time of passengers. The formulated discrete Markov system has over 10^{22} states even in the most simplified version. They also used a neural network and the final performance was slightly better than the best known algorithm and twice as good as the policy most popular in real elevator systems. Other successful RL/NDP applications in this field include large-scale job-shop scheduling [105,104,106], cell-phone channel allocation [76], manufacturing [47], and finance applications [58].

4.2. Robot learning

Robot learning is a difficult task in that it generally involves continuous state and action spaces, similar to process control problems. Barto *et al.* [11] proposed a learning structure for controlling a cart-pole system (inverted pendulum) that consisted of an associative search system and an adaptive critic system. Anderson [4] extended this work by training ANNs that learned to balance a pendulum given the actual state variables of the inverted pendulum as input with state space quantization for the evaluation network.

Schaal and Atkeson [74] used a nonparametric learning technique to learn the dynamics of a two-armed robot that juggles a device known as "devil-stick." They used task-specific knowledge to create an appropriate state space for learning. After 40 training runs, a policy capable of sustaining the juggling motion up to 100 hits was successfully obtained. A nonparametric approach was implemented to generalize the learning to unvisited states in the algorithm. This work was later extended to learn a pendulum swing-up task by using human demonstrations [8,73]. In the work, however, neither parametric nor nonparametric approach could learn a task of balancing the pendulum reliably due to poor parametrization and insufficient information for important regions of the state space, respectively.

We note that most robots used in assembly and manufacturing lines are trained in such a way that a human guides the robot through a sequence of motions that are memorized and simply replayed. Mahadevan and Connell [46] suggested a Q-learning algorithm with a clustering method for tabular approach to training a robot performing a box-pushing task. The robot learned to perform better than a human-programmed solution when a decomposition of sub-tasks was done carefully. Lin [45] used an ANN-based RL scheme to learn a simple navigation task. Asada *et al.* [6] designed a robot soccer control algorithm with a discretized state space based on some domain knowledge. Whereas most robot learning algorithms discretized the state space [87], Smart and Kaelbling [78] suggested an algorithm that deals with continuous state space in a more natural way. The main features are that approximated Q-values are used for training *neighboring* Q-values, and that a hyper-elliptic hull is designed to prevent extrapolation.

4.3. Process control

After the Bellman's publication, some efforts were made to use DP to solve various deterministic and stochastic optimal control problems. However, only a

few important results could be achieved through analytical solution, the most celebrated being the LQ optimal controller [103]. This combined with the limited computing power available at the time caused most control researchers to abandon the approach. As the computing power grew rapidly in the 1980s, some researchers used DP to solve simple stochastic optimal control problems, e.g., the dual adaptive control problem for a linear integrating system with an unknown gain [7].

While the developments in the AI and OR communities went largely unnoticed by the process control community, there were a few attempts for using similar techniques on process control problems. Hoskins and Himmelblau [31] first applied the RL concept to develop a learning control algorithm for a nonlinear CSTR but without any quantitative control objective function. They employed an adaptive heuristic critic algorithm suggested by Anderson [4] to train a neural network that maps the current state of the process to a suitable control action through *on-line* learning by experience. This approach uses qualitative subgoals for the controller and could closely approximate the behavior of the PID controller, but generalization of the method requires sufficient on-line experiments to cover the domain of interest at the cost of more trials for learning. Miller and Williams [51] used a temporal-difference learning scheme for control of a bio-reactor. They used a backpropagation network to estimate Q-values, and the internal state of a plant model was assumed to be known. The learning was based on trial-and-errors, and the search space was small (only 2 states).

Wilson and Martinez [102] studied batch process automation using fuzzy modeling and RL. To reduce the high dimensionality of the state and action space, they used a fuzzy look-up table for Q-values. Anderson *et al.* [5] suggested a RL method for tuning a PI controller of simulated heating coil. Their action space had only 9 discrete values, and therefore the look-up table method could be used. Martinez [50] suggested batch process optimization using RL, which was formulated as a two dimensional search space problem by shrinking the region of policy parameters. The work did not solve the DP, but only used a RL-based approach for exploring in the action space. Ahamed *et al.* [1] solved a power system control problem, which they represented it as a Markov chain with known transition probabilities so that the system dynamics would have finite candidate state and action sets for exploration and optimization.

Recently, Lee and co-workers have started introducing the concept of RL and NDP to the process control community and developed value/policy iteration-based algorithms to solve complex nonlinear process control problems. They include chemical/biochemical reactor control problems [43,35], a dual adaptive control problem [41], and a polymerization reactor control problem [40].

5. ISSUES IN APPLYING APPROXIMATE DP SCHEMES TO PROCESS CONTROL PROBLEMS

RL approaches in robot learning have dealt with continuous variables either by discretization or by function approximation, but they are based on trial-and-error on-line learning. For example, human controls a robot randomly to explore the state space in the early phase. They also assume that the environment does not change, which reduces the dimension of a concerned state space. On the other hand, NDP is more of an off-line based learning [18,14], and its basic assumption is that large amounts of data can be collected from simulation trajectories obtained with "good" suboptimal policies. Their common update rule, however, is based on the incremental TD-learning, which is difficult to apply to continuous state variables. In addition, complex dynamics of most chemical processes would limit the amount of data, whereas the NDP or related algorithms require huge amounts of data [18].

Despite various approximate methods from the RL/NDP communities, their applicability to process control problems is limited due to the following disparities:

1) Continuous state and action space: Infinite number of state and action values is common in process control problems due to their continuous nature. Furthermore, the number of state variables is generally large. In this case, discretization and the common "incremental" update rule are not practical approaches. Function approximation should also be used with caution, because approximation errors can grow quickly.
2) Costly on-line learning: Real-world-experience-based learning, which is the most prevalent approach in RL, is costly and risky for process control problems. For example, one cannot operate a chemical reactor in a random fashion without any suitable guidelines to explore the state space and gather data. Consequently, off-line learning using simulation trajectories should be preferred to on-line learning. Furthermore, one should also exercise a caution in implementing on-line control by insuring against "unreliable" optimal control actions calculated from only a partially learned cost-to-go function.
3) Limited data quantity: Though large amounts of simulation data can be collected for off-line learning, complex dynamics of most chemical processes still limit the state space that can be explored, leading to regions of sparse data. This limits the range over which the learned cost-to-go

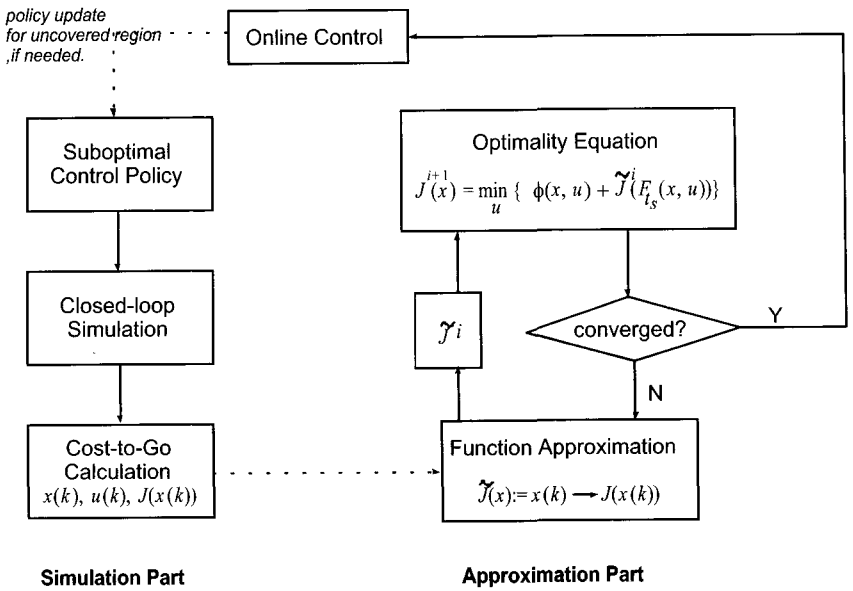


Fig. 3. Approximate value iteration scheme.

function is valid. Thus, learning and using of the cost-to-go function should be done cautiously by guarding against unreasonable extrapolations.

In summary, an adequate ADP approach for process control problems should be able to provide reliable control policies despite limited coverage of continuous state and action spaces by training data. One plausible approach is to simulate the system with a set of known suboptimal policies and generate trajectories to identify some superset of the portions of the state space pertinent to optimal control. In the absence of a model, an empirical model can be constructed, if necessary, and then value or policy iteration can be performed on the sampled data. A function approximator should also be designed to estimate cost-to-go values for points not visited by the simulations [35,43]. An approximate value iteration scheme is depicted in Fig. 3. In the following sections, issues concerning the suggested approach and the related work reported in the literature are discussed.

5.1. Generalization of Cost-to-Go function

5.1.1 Choice of approximator

All the algorithms described in section 3 assume that states and actions are finite sets and their sizes are manageable. When this is not true, generalization of cost-to-go values over the state space (or the state-action space for Q-learning) through function approximation may be the only recourse. One potential problem in using a function approximator in solving the Bellman equation is that small approximation errors can grow rapidly through the iteration to render the learned cost-to-go function useless. The minimization operator can bias the estimate significantly, and no prior knowledge on the structure of the cost-to-go function is available in general.

Typical approaches in the NDP and RL literature is to use a global approximator like a neural network to fit a cost-to-go function to the data. While it has seen some notable successes in problems such as the Tesauro's backgammon player [88,89,90] and the job-shop scheduling problem [105], there are many other less successful applications reported in the literature [21,75]. Successful implementations using localized networks like CMACs [2,3], radial basis functions, and memory-based learning methods [29] have also been reported [53,72,84].

The failure of the approaches using a general function approximator was first explained by Thrun and Schwartz [92] with what they called an "overestimation" effect. They assumed uniformly distributed and independent error in the approximation and derived the bounds on the necessary accuracy of the function approximator and discount factor. Sabes [69] showed that bias in optimality can be large when a basis function approximator is employed. Boyan and Moore [20] listed several simple simulation examples where popular approximators fail miserably. They used a model of the task dynamics and applied full DP backups off-line to a fixed set of states. Sutton [84] used the same examples and made them work by modifying the experimental setup. Sutton used an on-line learning scheme without a model for the state trajectories obtained from randomly sampled initial points. In summary, experiments with different function approximation schemes have produced mixed results, probably because of the different

learning schemes and problem setups.

Gordon [28] presented a 'stable' cost-to-go learning scheme with off-line iteration for a fixed set of states. A class of function approximators with a 'nonexpansion' property is shown to guarantee off-line convergence of cost-to-go values to some values. These function approximators do not exaggerate the differences between two cost-to-go functions over iterations in the sense of infinity norm. That is, if we have two functions f and g, their approximations from this class of approximators, $\tilde{f}$ and $\tilde{g}$, satisfy

$$\left|\tilde{f}(x)-\tilde{g}(x)\right| \le \left\|f(x)-g(x)\right\|_{\infty}. \tag{26}$$

The class includes the k-nearest neighbor, kernel-based approximators, and other type of "averagers" having the following structure:

$$\tilde{f}(x)=\beta_0+\sum_{i=1}^{k}\beta_i f(x_i) \tag{27}$$

with

$$\sum_{i=0}^{k}\beta_i=1 \quad \text{and} \quad \beta_i \ge 0. \tag{28}$$

If the off-line learning using a value or policy iteration scheme is chosen for the cost-to-go learning, memory-based approximators with the nonexpansion property show better performance than global parametric approximators [20,40].

5.1.2 Control of extrapolation

Even though iteration error is monotonically decreased and convergence is guaranteed with a proper choice of approximator, sparse data in a high-dimensional state space can result in poor control due to the limited validity of the learned cost-to-go function [35,40]. The cost-to-go approximator may not be valid in regions where no training data is available. Hence, control of extrapolation is a critical issue in using the approximated cost-to-go function both during off-line value/policy iteration and during on-line control. This issue has not been studied explicitly in the RL/NDP literature because of the characteristics of their problems (e.g. trial-and-error learning and huge data).

Related work dealing with excessive extrapolation is found in the robot learning literature [78]. They construct an approximate convex hull, which is called "independent variable hull (IVH)" taking an elliptic form as depicted in Fig. 4. Whenever one has to estimate the cost-to-go for a query point, the IVH is calculated around the training data that lie closer than some threshold value from the query point. Any queries within the convex hull are considered to be reliable and those outside are deemed unreliable. This approach is not computationally attractive. It also gives up making a prediction for the point outside the hull and requires more random explorations. In addition, the use of the convex hull may be misleading if the elliptic hull contains a significant empty region around the query point.

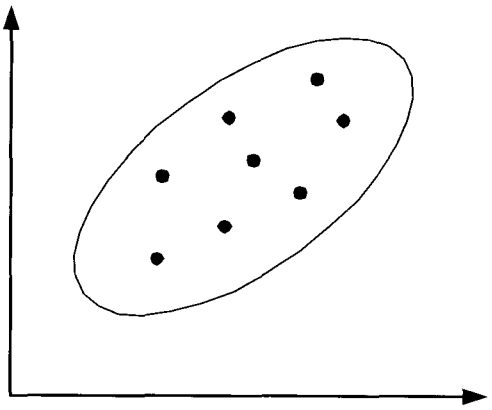


Fig. 4. Two-dimensional independent variable hull (IVH).

One logical approach to this problem is to estimate local density of training data points around the query point and use it as a reliability indicator. One such idea was reported by Leonard *et al.* [44] for design of a reliable neural network structure by using a radial basis function and a nonparametric probability density estimator [61]. Lee and Lee [42] suggested a penalty function-based approach, which adjusts the estimate of cost-to-go according to the local data density.

5.2. Solution property: convergence and optimality

Understanding the accuracy of a learned cost-to-go function and its corresponding policy is very important for successful implementation. Researchers have been interested in understanding the convergence property of a learning algorithm and its error bound (or bias from the "true" optimal cost-to-go function). Though exact value and policy iterations are shown to converge and their error bounds are presented in standard DP textbooks [65], most of the approximate DP algorithms employing function approximation are yet to be understood fully at a theoretical level. This is particularly true for problems with continuous state-action spaces.

Gordon's value iteration algorithm using a local averager with the nonexpansion property [28] is convergent but its error bound is only available for the 1-nearest neighborhood estimator. Tsitsiklis and Van Roy [94] provided a proof of convergence and its accuracy for linear function approximators when applied to finite MDPs with temporal difference

learning under a particular on-line state sampling scheme. They concluded that the convergence properties of general nonlinear function approximators (e.g. neural network) were still unclear.

Ormoneit and Sen [60] suggested a kernel-based Q-learning for continuous state space using sample trajectories only. The algorithm is designed for discounted infinite horizon cost and employs a kernel averager like Gordon's to average a collection of sampled data where a *specific* action was applied. Hence, a separate training data set exists marked with each action to approximate the Q-function. This way they show that the kernel-based Q-learning can converge to a unique solution, and the optimal solution can be obtained as the number of samples increases to infinity, which they call *consistency*. They also conclude that all reinforcement learning using finite samples is subject to bias. Same results for average cost problems are provided in [59]. Though the theoretical argument on the convergence property could be established, error bounds for practical set-up of the algorithm are yet to be provided.

Sutton *et al.* [79] suggested an alternative approach that directly optimizes over the policy space. The algorithm uses a parametric representation of a policy, and gradient-based optimization is performed to update the parameter set. As the number of parameters increases, the learning converges to an optimal policy in a "local" sense due to the gradient-based search. Konda and Tsitsiklis [37] proposed a similar approach under an actor-critic framework, which guarantees convergence to a locally optimal policy. In both approaches, they consider a finite MDP with a randomized stationary policy that gives action selection probabilities.

De Farias and Van Roy [25] proposed an approximate LP approach to solving DP based on parameterized approximation. They derived an error bound that characterizes the quality of approximations compared to the "best possible" approximation of the optimal cost-to-go function with given basis functions. The approach is, however, difficult to generalize to continuous state problems, because the LP approach requires a description of the system's stochastic behavior as finite number of constraints, which is impossible without discretization of a probabilistic model.

6. FUTURE DIRECTIONS AND CONCLUDING REMARKS

Though there exist many RL/NDP methodologies for solving DP in an approximate sense, only a limited number of them are applicable to process control problems given the problems' nature. We are of the opinion that finding a fixed-solution of the Bellman equation using value/policy iteration with a stable function approximator (e.g. local averager) is the best strategy. In general, solving DP by RL/NDP methods with function approximation remains more a problem-specific art than a generalized science.

In order to solve highly complex problems with guaranteed performance, the following questions should be addressed.

- Reliable use of cost-to-go approximation: Though we can control undue extrapolations by using some penalty terms, function approximation can be carried out in a more systematic way if an error bound can be derived for a general class of problems. This analysis is yet to be reported for continuous problems.
- Dealing with large action space: The suggested algorithms can suffer from the same curse-of-dimensionality as the dimension of action space increases. This is because the action space should be searched over for finding a greedy control action.
- Applicability of policy space algorithms: All the methods and issues described above are mainly concerned with approximating the cost-to-go function aimed at solving the Bellman equation directly. Then the learned cost-to-go function is used to prescribe a near-optimal policy. A new approach recently advocated is to approximate and optimize directly over the policy space, which is called *policy-gradient method* [37,79,49]. The method was motivated by the disadvantage of the cost-to-go function based approach that can result very different actions for "close" states from the greedy policy in the presence of approximation errors. This can be avoided by controlling the smoothness of the policy directly. However, the algorithms still need significant investigation to be recast as an applicable framework for process control problems.

REFERENCES

[1] T. P. I. Ahamed, P. S. N. Rao, and P. S. Sastry, "A reinforcement learning approach to automatic generation control," *Electric Power Systems Research*, vol. 63, no. 1, pp. 9-26, 2002.

[2] J. S. Albus, "Data storage in the cerebellar model articulation controller," *Journal of Dynamic Systems, Measurement and Control*, pp. 228-233, 1975.

[3] J. S. Albus, "A new approach to manipulator control: The cerebellar model articulation controller (CMAC)," *Journal of Dynamic Systems, Measurement and Control*, pp. 220-227, 1975.

[4] C. W. Anderson, "Learning to control an inverted pendulum using neural networks," *IEEE Control Systems Magazine*, vol. 9, no. 3, pp. 31-37, 1989.

[5] C. W. Anderson, D. C. Hittle, A. D. Katz, and R. M. Kretchmar, "Synthesis of reinforcement

learning, neural networks and PI control applied to a simulated heating coil," *Artificial Intelligence in Engineering*, vol. 11, no. 4, pp. 421-429, 1997.

[6] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda, "Purposive behavior acquisition for a real robot by vision-based reinforcement learning," *Machine Learning*, vol. 23, pp. 279-303, 1996.

[7] K. J. Åström and A. Helmersson, "Dual control of an integrator with unknown gain," *Comp. & Maths. with Appls.*, vol. 12A, pp. 653-662, 1986.

[8] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," *Proc. of the Fourteenth International Conference on Machine Learning*, pp. 12-20, San Francisco, CA, 1997.

[9] L. Baird III, "Residual algorithms: Reinforcement learning with function approximation," *Proc. of the International Conference on Machine Learning*, pp. 30-37, 1995.

[10] A. G. Barto, S. J. Bradtke, and S. P. Singh, "Learning to act using real-time dynamic programming," *Artificial Intelligence*, vol. 72, no. 1, pp. 81-138, 1995.

[11] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 13, no. 5, pp. 834-846, 1983.

[12] R. E. Bellman, *Dynamic Programming,* Princeton University Press, New Jersey, 1957.

[13] D. P. Bertsekas. *Dynamic Programming and Optimal Control,* Athena Scientic, Belmont, MA, 2nd edition, 2000.

[14] D. P. Bertsekas, "Neuro-dynamic programming: An overview," In J. B. Rawlings, B. A. Ogunnaike, and J. W. Eaton, editors, *Proc. of Sixth International Conference on Chemical Process Control*, 2001.

[15] D. P. Bertsekas and D. A. Castañon, "Adaptive aggregation for infinite horizon dynamic programming," *IEEE Trans. on Automatic Control*, vol. 34, no. 6, pp. 589-598, 1989.

[16] D. P. Bertsekas and R. G. Gallager, *Data Networks,* Prentice-Hall, Englewood Cliffs, NJ, 2nd edition, 1992.

[17] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods,* Prentice-Hall, Englewood Cliffs, NJ, 1989.

[18] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming,* Athena Scientic, Belmont, Massachusetts, 1996.

[19] V. Borkar, "A convex analytic approach to Markov decision processes," *Probability Theory and Related Fields*, vol. 78, pp. 583-602, 1988.

[20] J. A. Boyan and A. W. Moore, "Generalization in reinforcement learning: safely approximating the value function," In G. Tesauro and D. Touretzky, editors, *Advances in Neural Information Processing Systems*, vol. 7, Morgan Kaufmann, 1995.

[21] S. J. Bradtke, "Reinforcement learning applied to linear quadratic regulation," In S. J. Hanson, J. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems*, vol. 5, Morgan Kaufmann, 1993.

[22] R. Crites and A. G. Barto, "Improving elevator performance using reinforcement learning," In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, vol. 8, MIT Press, San Francisco, CA, 1996.

[23] R. Crites and A. G. Barto, "Elevator group control using multiple reinforcement learning agents," *Machine Learning*, vol. 33, pp. 235-262, 1998.

[24] P. Dayan. "The convergence of TD(λ) for general λ," *Machine Learning*, vol. 8, pp. 341-362, 1992.

[25] D. P. de Farias and B. Van Roy, "The linear programming approach to approximate dynamic programming," *Operations Research*, vol. 51, no. 6, pp. 850-865, 2003.

[26] E. V. Denardo, "On linear programming in a Markov decision problem," *Management Science*, vol. 16, pp. 282-288, 1970.

[27] C. G. Atkeson and J. Santamaria, "A comparison of direct and model-based reinforcement learning," *Proc. of the International Conference on Robotics and Automation*, 1997.

[28] G. J. Gordon, "Stable function approximation in dynamic programming," *Proc. of the Twelfth International Conference on Machine Learning*, San Francisco, CA, pp. 261-268, 1995.

[29] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction,* Springer-Verlag, New York, NY, 2001.

[30] A. Hordijk and L. C. M. Kallenberg, "Linear programming and Markov decision chains," *Management Science*, vol. 25, pp. 352-362, 1979.

[31] J. C. Hoskins and D. M. Himmelblau, "Process control via artificial neural networks and reinforcement learning," *Computers & Chemical Engineering*, vol. 16, no. 4, pp. 241-251, 1992.

[32] R. A. Howard, *Dynamic Programming and Markov Processes,* MIT Press, Cambridge, MA, 1960.

[33] T. Jaakkola, M. I. Jordan, and S. P. Singh, "On the convergence of stochastic iterative dynamic programming algorithms," *Neural Computation*, vol. 6, no. 6, pp. 1185-1201, 1994.

[34] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey,"

Journal of Artificial Intelligence Research, vol. 4, pp. 237-285, 1996.

[35] N. S. Kaisare, J. M. Lee, and J. H. Lee, "Simulation based strategy for nonlinear optimal control: Application to a microbial cell reactor," *International Journal of Robust and Nonlinear Control*, vol. 13, no. 3-4, pp. 347-363, 2002.

[36] S. Koenig and R. G. Simmons, "Complexity analysis of real-time reinforcement learning," *Proc. of the Eleventh National Conference on Artificial Intelligence*, Menlo Park, CA, pp. 99-105, 1993.

[37] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in neural information processing systems*, vol. 12, 2000.

[38] B. J. A. Kröse and J. W. M. van Dam, "Adaptive state space quantisation for reinforcement learning of collision-free navigation," *Proc. of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Piscataway, NJ, 1992.

[39] P. R. Kumar and P. P. Varaiya, *Stochastic Systems: Estimation, Identification, and Adaptive Control,* Prentice Hall, Englewood Cliffs, NJ, 1986.

[40] J. M. Lee, N. S. Kaisare, and J. H. Lee, "Simulation-based dynamic programming strategy for improvement of control policies," *AIChE Annual Meeting*, San Francisco, CA, paper 438c, 2003.

[41] J. M. Lee and J. H. Lee, "Neuro-dynamic programming approach to dual control problem," *AIChE Annual Meeting*, Reno, NV, paper 276e, 2001.

[42] J. M. Lee and J. H. Lee, "Approximate dynamic programming based approaches for input-output data-driven control of nonlinear processes," *Automatica*, 2004. Submitted.

[43] J. M. Lee and J. H. Lee, "Simulation-based learning of cost-to-go for control of nonlinear processes," *Korean J. Chem. Eng.*, vol. 21, no. 2, pp. 338-344, 2004.

[44] J. A. Leonard, M. A. Kramer, and L. H. Ungar, "A neural network architecture that computes its own reliability," *Computers & Chemical Engineering*, vol. 16, pp. 819-835, 1992.

[45] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, plannin and teaching." *Machine Learning*, vol. 8, pp. 293-321, 1992.

[46] S. Mahadevan and J. Connell, "Automatic programming of behavior-based robots using rein-forcement learning," *Machine Learning*, vol. 55, no. 2-3, pp. 311-365, 1992.

[47] S. Mahadevan, N. Marchalleck, T. K. Das, and A. Gosavi, "Self-improving factory simulation using continuous-time average-reward reinforcement learning," *Proc. of 14th International Conference on Machine Learning*, pp. 202-210, 1997.

[48] A. S. Manne, "Linear programming and sequential decisions," *Management Science*, vol. 6, no. 3, pp. 259-267, 1960.

[49] P. Marbach and J. N. Tsitsiklis, "Simulation-based optimization of Markov reward processes," *IEEE Trans. on Automatic Control*, vol. 46, no. 2, pp. 191-209, 2001.

[50] E. C. Martinez, "Batch process modeling for optimization using reinforcement learning," *Computers & Chemical Engineering*, vol. 24, pp. 1187-1193, 2000.

[51] S. Miller and R. J. Williams, "Temporal difference learning: A chemical process control application," In A. F. Murray, editor, *Applications of Artificial Neural Networks,* Kluwer, Norwell, MA, 1995.

[52] A. Moore and C. Atkeson, "The parti-game algorithm for variable resolution reinforcement learning in multidimensional state spaces. *Machine Learning*, vol. 21, no. 3, pp. 199-233, 1995.

[53] A. W. Moore, *Efficient Memory Based Robot Learning,* PhD thesis, Cambridge University, October 1991.

[54] A. W. Moore and C. G. Atkeson, "Prioritized sweeping: Reinforcement learning with less data and less time," *Machine Learning*, vol. 13, pp. 103-130, 1993.

[55] M. Morari and J. H. Lee, "Model predictive control: Past, present and future," *Computers & Chemical Engineering*, vol. 23, pp. 667-682, 1999.

[56] R. Munos, "A convergent reinforcement learning algorithm in the continuous case based on a finite difference method," *Proc. of the International Joint Conference on Artificial Intelligence*, 1997.

[57] R. Munos, "A study of reinforcement learning in the continuous case by means of viscosity solutions," *Machine Learning Journal*, vol. 40, pp. 265-299, 2000.

[58] R. Neuneier, "Enhancing Q-learning for optimal asset allocation," In M. Jordan, M. Kearns, and S. Solla, editors, *Advances in Neural Information Processing Systems*, vol. 10, 1997.

[59] D. Ormoneit and P. W. Glynn, "Kernel-based reinforcement learning in average-cost problems," *IEEE Trans. on Automatic Control*, vol. 47, no. 10, pp. 1624-1636, 2002.

[60] D. Ormoneit and S. Sen, "Kernel-based reinforcement learning," *Machine Learning*, vol. 49, pp. 161-178, 2002.

[61] E. Parzen, "On estimation of a probability

density function and mode," *Ann. Math. Statist.*, vol. 33, pp. 1065-1076, 1962.

[62] J. Peng, *Efficient Dynamic Programming-Based Learning for Control,* PhD thesis, North-eastern University, Boston, MA, 1993.

[63] J. Peng and R. J. Williams, "Efficient learning and planning within the Dyna framework," *Adaptive Behavior*, vol. 1, no. 4. pp. 437-454, 1993.

[64] D. V. Prokhorov and D. C. Wunsch II, "Adaptive critic designs," *IEEE Trans. on Neural Networks*, vol. 8, no. 5, pp. 997-1007, September 1997.

[65] M. L. Puterman, *Markov Decision Processes,* Wiley, New York, NY, 1994.

[66] S. J. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control Engineering Practice*, vol. 11, no. 7, pp. 733-764, 2003.

[67] U. Rüde, *Mathematical and Computational Techniques for Multilevel Adaptive Methods,* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1993.

[68] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*, Technical Report CUED/F-INFENG/TR 166, Engineering Department, Cambridge University, 1994.

[69] P. Sabes, "Approximating Q-values with basis function representations," *Proc. of the Fourth Connectionist Models Summer School*, Hillsdale, NJ, 1993.

[70] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM J. Res. Develop.*, pp. 210-229, 1959.

[71] A. L. Samuel, "Some studies in machine learning using the game of checkers II - recent progress," *IBM J. Res. Develop.*, pp. 601-617, 1967.

[72] J. C. Santamaría, R. S. Sutton, and A. Ram, "Experiments with reinforcement learning in problems with continuous state and action spaces," *Adaptive Behavior*, vol. 6, no. 2, pp. 163-217, 1997.

[73] S. Schaal, "Learning from demonstration," In M. C. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, vol. 9, pp. 1040-1046, 1997.

[74] S. Schaal and C. Atkeson, "Robot juggling: An implementation of memory-based learning," *IEEE Control Systems*, vol. 14, no. 1, pp. 57-71, 1994.

[75] N. N. Schraudolph, P. Dayan, and T. J. Sejnowski, "Temporal difference learning of position evaluation in the game of Go," In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, vol. 6, pp. 817-824, 1994.

[76] S. Singh and D. Bertsekas, "Reinforcement learning for dynamic channel allocation in cellular telephone systems," In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, vol. 9, pp. 974-980, 1997.

[77] S. P. Singh and R. S. Sutton, "Reinforcement learning with replacing eligibility traces," *Machine Learning*, vol. 22, pp. 123-158, 1996.

[78] W. D. Smart and L. P. Kaelbling, "Practical reinforcement learning in continuous spaces," *Proc. 17th International Conf. on Machine Learning*, pp. 903-910, 2000.

[79] R. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforce-ment learning with function approximation," In S. A. Solla, T. K. Leen, and K.-R. Muller, editors, *Advances in Neural Information Processing Systems*, vol. 12, pp. 1057-1063, 2000.

[80] R. S. Sutton, *Temporal Credit Assignment in Reinforcement Learning,* PhD thesis, University of Massachusetts, Amherst, MA, 1984.

[81] R. S. Sutton, "Learning to predict by the method of temporal differences," *Machine Learning*, vol. 3. no. 1, pp. 9-44, 1988.

[82] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," *Proc. of the Seventh International Conference on Machine Learning*, Austin, TX, 1990.

[83] R. S. Sutton, "Planning by incremental dynamic programming," *Proc. of the Eighth International Workshop on Machine Learning*, pp. 353-357, 1991.

[84] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, vol. 8, pp. 1038-1044, 1996.

[85] R. S. Sutton and A. G. Barto, "Toward a modern theory of adaptive networks: Expectation and prediction," *Psycol. Rev.*, vol. 88, no. 2, pp. 135-170, 1981.

[86] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction,* MIT Press, Cambridge, MA, 1998.

[87] M. Takeda, T. Nakamura, M. Imai, T. Ogasawara, and M. Asada, "Enhanced continuous valued Q-learning for real autonomous robots," *Advanced Robotics*, vol. 14, no. 5, pp. 439-442, 2000.

[88] G. Tesauro, "Practical issues in temporal difference learning," *Machine Learning*, vol. 8, pp. 257-277, 1992.

[89] G. Tesauro, "TD-Gammon, a self-teaching

backgammon program, achieves master-level play," *Neural Computation*, vol. 6, no. 2, pp. 215-219, 1994.

[90] G. Tesauro, "Temporal difference learning and TD-Gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58-67, 1995.

[91] S. Thrun, "Learning to play the game of chess," In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems*, vol. 7, 1995.

[92] S. Thrun and A. Schwartz, "Issues in using function approximation for reinforcement learning," *Proc. of the Fourth Connectionist Models Summer School*, Hillsdale, NJ, 1993.

[93] J. N. Tsitsiklis, "Asynchronous stochastic approximation and Q-learning," *Machine Learning*, vol. 16, pp. 185-202, 1994.

[94] J. N. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Trans. on Automatic Control*, vol. 42, no. 5, pp. 674-690, 1997.

[95] B. Van Roy, "Neuro-dynamic programming: Overview and recent trends," In E. Feinberg and A. Shwartz, editors, *Handbook of Markov Decision Processes: Methods and Applications*, Kluwer, Boston, MA, 2001.

[96] C. J. C. H. Watkins, *Learning from Delayed Rewards*, PhD thesis, University of Cambridge, England, 1989.

[97] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279-292, 1992.

[98] P. J. Werbos, "Advanced forecasting methods for global crisis warning and models of intelligence," *General Systems Yearbook*, vol. 22, pp. 25-38, 1977.

[99] P. J. Werbos, "Approximate dynamic programming for real-time control and neural modeling," In D. A. White and D. A. Sofge, editors, *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, Van Nostrand Reinhold, New York, pp. 493-525, 1992.

[100] S. D. Whitehead, "Complexity and cooperation in Q-learning," *Proc. of the Eighth International Workshop on Machine Learning*, Evanston, IL, 1991.

[101] R. J. Williams and L. C. Baird III, "Analysis of some incremental variants of policy iteration: First steps toward understanding actor-critic learning systems," *Technical Report NU-CCS-93-14*, Northeastern University, College of Computer Science, Boston, MA, 1993.

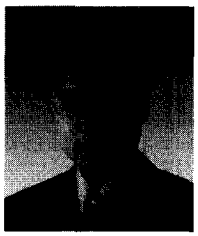
[102] J. A. Wilson and E. C. Martinez, "Neuro-fuzzy modeling and control of a batch process involving simultaneous reaction and distillation," *Computers & Chemical Engineering*, vol. 21S, pp. S1233-S1238, 1997.

[103] M. Wonham, "Stochastic control problems," In B. Friedland, editor, *Stochastic Problems in Control*, ASME, New York, 1968.

[104] W. Zhang, *Reinforcement Learning for Job-Shop Scheduling*, PhD thesis, Oregon State University, 1996. Also available as Technical Report CS-96-30-1.

[105] W. Zhang and T. G. Dietterich, "A reinforcement learning approach to job-shop scheduling," *Proc. of the Twelfth International Conference on Machine Learning*, San Francisco, CA, pp. 1114-1120, 1995.

[106] W. Zhang and T. G. Dietterich, "High-performance job-shop scheduling with a time-delay TD(λ) network," In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, vol. 8, 1996.



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 post doctoral 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 California Institute of Technology, Pasadena, in 1991. From 1991 to 1998, he was with the Department of Chemical Engineering at Auburn University, AL, as an Assistant Professor and an Associate Professor. From 1998 to 2000, he was with 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 Center for Process Systems Engineering at 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.