Approximate Reinforcement Learning: An Overview

Lucian Buşoniu*, Damien Ernst†, Bart De Schutter*, Robert Babuška*
*Delft Center for Systems & Control, Delft Univ. of Technology, Netherlands; {i.l.busoniu,b.deschutter,r.babuska}@tudelft.nl
†Research Associate of the FRS-FNRS; Systems and Modeling Unit, University of Liège, Belgium; dernst@ulg.ac.be

***Abstract*—Reinforcement learning (RL) allows agents to learn how to optimally interact with complex environments. Fueled by recent advances in approximation-based algorithms, RL has obtained impressive successes in robotics, artificial intelligence, control, operations research, etc. However, the scarcity of survey papers about approximate RL makes it difficult for newcomers to grasp this intricate field. With the present overview, we take a step toward alleviating this situation. We review methods for approximate RL, starting from their dynamic programming roots and organizing them into three major classes: approximate value iteration, policy iteration, and policy search. Each class is subdivided into representative categories, highlighting among others offline and online algorithms, policy gradient methods, and simulation-based techniques. We also compare the different categories of methods, and outline possible ways to enhance the reviewed algorithms.**

***Index Terms*—reinforcement learning, function approximation, value iteration, policy iteration, policy search.**

I. Introduction

Using reinforcement learning (RL), agents (controllers) can learn how to optimally interact with complex environments (systems). RL can address problems from a variety of domains, including artificial intelligence (AI), robotics, automatic control, operations research, economics, medicine, etc. For instance, in AI, RL provides a way to build learning agents that optimize their behavior in initially unknown environments [1]–[3]. In automatic control, RL can in principle solve nonlinear and stochastic optimal control problems, without requiring a model [4]–[7]. An RL agent interacts with its environment by measuring states and applying actions according to its policy. A reward signal indicates the immediate performance, and the goal is to find an optimal policy that maximizes the value function, i.e., the cumulative long-term reward as a function of the states and possibly of the actions.

Classical RL requires exact representations of value functions and policies, and is therefore limited to small, discrete problems. However, most realistic problems have variables with a large or infinite number of possible values (e.g., continuous variables); in such problems, value functions and policies must be *approximated*. Fueled by recent advances in approximation-based algorithms, RL has obtained impressive successes in applications such as robot control [8], autonomous helicopter flight [9], interfacing an animal brain with a robot arm [10], simulated treatment of HIV infections [11], etc.

Unfortunately, a newcomer to the field of *approximate RL* will find it difficult to grasp, due to the intricate and extensive landscape of algorithms and results, coupled with the scarcity of survey papers[1]. In this overview, we take a step toward alleviating this situation, by painting a high-level picture of the field, and going in-depth into representative approaches. Starting from their exact dynamic programming (DP) and RL roots, we describe three major classes of approximate RL techniques: approximate value iteration, policy iteration, and policy search. We introduce offline algorithms, as well as online algorithms, which collect their own data by interaction with the system. We pay attention to policy gradient and actor-critic methods, as well as to simulation-based policy iteration. In closing, the various types of approaches are compared, several important ways to enhance approximate RL methods are outlined, and pointers to further reading are provided.

Next, Sec. II introduces classical, exact DP and RL. Then, Sec. III discusses the need for approximation in RL and a taxonomy of approximate RL methods. Sections IV – VI present, in turn, approximate value iteration, policy iteration, and policy search. Sec. VII closes the paper with a discussion and an outlook.

II. Preliminaries. Classical DP and RL

This section briefly introduces the RL problem, together with classical algorithms to solve it, in the framework of Markov decision processes.

Consider a Markov decision process with state space X and action space U. For now, X and U are assumed countable, but under appropriate technical conditions, the formalism can be extended to uncountable sets [12]. The probability that the next state x_{k+1} is reached after action u_k is taken in state x_k is $f(x_k, u_k, x_{k+1})$, where $f : X \times U \times X \to [0, 1]$ is the transition probability function. After the transition to x_{k+1}, a reward $r_{k+1} = \rho(x_k, u_k, x_{k+1})$ is received, where $\rho : X \times U \times X \to \mathbb{R}$ is the reward function. The symbol k denotes the discrete time index.[2] The expected infinite-horizon discounted return of initial state x_0 under a policy $h : X \to U$ is:

$$R^h(x_0) = \lim_{K \to \infty} \mathrm{E}_{x_{k+1} \sim f(x_k, h(x_k), \cdot)} \left\{ \sum_{k=0}^{K} \gamma^k r_{k+1} \right\} \quad (1)$$

where $r_{k+1} = \rho(x_k, u_k, x_{k+1})$, $\gamma \in (0, 1)$ is the discount factor, and the notation $x_{k+1} \sim f(x_k, h(x_k), \cdot)$ means that x_{k+1} is drawn from the distribution $f(x_k, h(x_k), \cdot)$. The goal is to find an optimal policy h^* that maximizes the return (1) from every $x_0 \in X$. Other types of return can also be used, such as finite-horizon or averaged over time [13], [6].

Many algorithms employ value functions to find h^*. The Q-function (state-action value function) of a policy h is the expected return when starting in a given state, applying a given action, and following h thereafter: $Q^h(x, u) = \mathrm{E}_{x' \sim f(x,u,\cdot)} \left\{ \rho(x, u, x') + \gamma R^h(x') \right\}$. The optimal Q-function

[1] See however Sec. VII, *Further reading*, for recent overview texts with an important approximate RL component.

[2] We use control-theoretical notation; operations-research and AI notation are also commonly employed in RL.

is the best that can be obtained by any policy: $Q^*(x,u) = \max_h Q^h(x,u)$. Any policy h^* that selects at each state an action with the largest optimal Q-value, i.e., that satisfies:

$$h^*(x) \in \arg\max_u Q^*(x,u) \tag{2}$$

is optimal (it maximizes the return). In general, a policy that chooses actions by maximizing a certain Q-function is called *greedy* in this Q-function. Policy Q-functions and, respectively, the optimal Q-function satisfy the Bellman equations:

$$Q^h(x,u) = \mathrm{E}_{x' \sim f(x,u,\cdot)} \left\{ \rho(x,u,x') + \gamma Q^h(x', h(x')) \right\} \tag{3}$$

$$Q^*(x,u) = \mathrm{E}_{x' \sim f(x,u,\cdot)} \left\{ \rho(x,u,x') + \gamma \max_{u'} Q^*(x',u') \right\} \tag{4}$$

State value functions can be used as an alternative to Q-functions. In particular, the V-function of policy h is $V^h = Q^h(x, h(x))$, and the optimal V-function is $V^*(x) = \max_u Q^*(x,u)$. Henceforth, we will prefer using Q-functions, because they make greedy policies easier to find – see (2) – than when V-functions are employed. Modulo this difficulty, many of the algorithms easily extend to the V-function case.

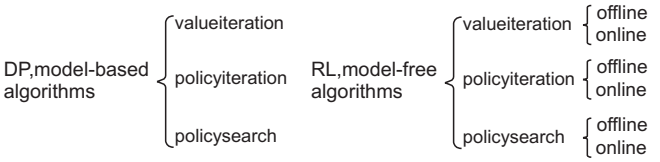

Fig. 1. A taxonomy of DP and RL algorithms.

Fig. 1 shows the algorithms taxonomy that we will employ. DP algorithms are model-based: they find h^* using knowledge of the model (the functions f and ρ).[3] RL algorithms are model-free: they find h^* from transition and reward data, without using f and ρ. Moreover, online RL algorithms collect their own data, by interacting with the system while they learn. Some RL algorithms are model-learning (not shown in Fig. 1): they estimate f and ρ, and apply DP techniques to these estimates.

Looking at the path they take to find an optimal policy, the algorithms can be classified along a second, orthogonal axis into three classes: value iteration, policy iteration, and policy search. We next introduce these classes, together with several central algorithms useful later in this survey.

Value iteration algorithms directly search for an optimal value function, which they use to compute an optimal policy. An important DP algorithm from this class is Q-iteration[4], which starts from some initial Q_0 and updates it at each iteration ℓ by turning the Bellman equation (4) into an assignment:

$$Q_{\ell+1}(x,u) = \mathrm{E}_{x' \sim f(x,u,\cdot)} \left\{ \rho(x,u,x') + \gamma \max_{u'} Q_\ell(x',u') \right\} \tag{5}$$

In the countable state-action spaces assumed, the expectation can be written as a sum. Q-iteration asymptotically converges to Q^*, which can then be used to obtain h^* with (2).

[3]DP is just one model-based paradigm for optimal control; there are many others, such as model-predictive control.

[4]Throughout the paper, sans-serif font is used for algorithm names.

The most widely used RL algorithm from the value iteration class is Q-learning [14], which updates the Q-function online, using observed state transitions and rewards, i.e., data tuples of the form $(x_k, u_k, x_{k+1}, r_{k+1})$:

$$Q_{k+1}(x_k,u_k) = Q_k(x_k,u_k) + \alpha_k [r_{k+1} + \gamma \max_{u'} Q_k(x_{k+1},u') - Q_k(x_k,u_k)] \tag{6}$$

where $\alpha_k \in (0,1]$ is the learning rate. Q-learning can be seen as a sample-based, stochastic approximation of Q-iteration. At each step, the estimate for the optimal Q-value of (x_k,u_k) is corrected by the 'temporal difference' between the update target $r_{k+1} + \gamma \max_{u'} Q_k(x_{k+1},u')$ – given by the Q-iteration update (5) or the Bellman equation (4) – and the current estimate $Q_k(x_k,u_k)$.

To converge to Q^*, Q-learning crucially requires *exploration*: the probability of attempting new, non-greedy actions must always be non-zero, to avoid becoming stuck in a local optimum. The tradeoff between exploration and greedy action choices ('exploitation') is essential for the performance of any online RL algorithm. A classical way to achieve this tradeoff is the ε-greedy policy, [1, Sec. 2.2]:

$$u_k = h_k(x_k) = \begin{cases} u \in \arg\max_{\bar{u}} Q_k(x_k,\bar{u}) & \text{w.p. } 1-\varepsilon_k \\ \text{a uniform random } u \in U & \text{w.p. } \varepsilon_k \end{cases} \tag{7}$$

where $\varepsilon_k \in (0,1)$ is the exploration probability at step k.

Policy iteration algorithms evaluate policies by constructing their value functions (*policy evaluation* step), and use these value functions to find greedy, improved policies (*policy improvement* step). For instance, a DP algorithm for policy iteration starts with some initial policy h_0, and at every iteration $\ell \geq 0$, finds Q^{h_ℓ} by iteratively applying the Bellman equation (3), in a similar way to Q-iteration. Then, an improved policy is determined:

$$h_{\ell+1}(x) \in \arg\max_u Q^{h_\ell}(x,u) \tag{8}$$

and the algorithm continues with this policy at the next iteration. This algorithm is guaranteed to converge to h^*, see [1, Sec. 4.3].

A well-known algorithm from the online RL, policy iteration class is SARSA [15]. SARSA employs tuples of the form $(x_k, u_k, r_{k+1}, x_{k+1}, u_{k+1})$ to update the Q-function:

$$Q_{k+1}(x_k,u_k) = Q_k(x_k,u_k) + \alpha_k [r_{k+1} + \gamma Q_k(x_{k+1},u_{k+1}) - Q_k(x_k,u_k)] \tag{9}$$

and applies an exploration-exploitation policy such as ε-greedy (7). In contrast to Q-learning, the update target is $r_{k+1} + \gamma Q_k(x_{k+1},u_{k+1})$. Since $u_{k+1} = h_{k+1}(x_{k+1})$, this target is derived from the Bellman equation (3), and under a fixed policy, SARSA would perform online policy evaluation. However, since the exploitative part of the policy is always greedy in Q_k, SARSA implicitly performs policy improvements, and is therefore a type of online policy iteration.

Policy search algorithms aim to find an optimal policy directly. To this end, they solve an optimization problem:

$h^* \in \arg\max_h s(h)$, where $s(h)$ is a score value representative for the returns obtained by h. We will delve into details about policy search in Sec. VI, in the approximate setting.

Relation to other taxonomies: Policy search is sometimes called 'approximation in policy space', and by contrast value and policy iteration are grouped together into 'approximation in value function space' [6]. Also, it may seem striking that we classify Q-learning as online value iteration and SARSA as online policy iteration, since traditionally value and policy iteration are offline, while Q-learning and SARSA are respectively 'off-policy' and 'on-policy' (on-policy means here that the Q-value $Q(x_{k+1}, u_{k+1})$ of the actually chosen next action u_{k+1} is used in the updates, while off-policy methods may use another action). However, as seen above, Q-learning is an online, incremental way of performing Q-iteration, and SARSA is an incremental policy iteration where policy improvements are 'hidden' in the greedy action selections. We find this deeper relationship more interesting; e.g., some policy iteration methods can use data collected *off-policy*!

Finally, we wish to emphasize the distinction between model-based and model-learning methods. While the latter are (somewhat confusingly) called 'model-based' in the AI literature, we reserve the name 'model-based' for methods that truly require an a priori model of the problem.

III. Approximation in RL

Classical RL and DP algorithms require exact representations of value functions Q, V or policies h. The only general way to guarantee exactness is to store distinct values for every state or state-action pair. Unfortunately, this is impossible when X, and possibly U, contain a very large or infinite number of elements. Instead, *function approximators* must be used to represent Q, V, or h. Combining approximators with RL leads to *approximate RL*, the main subject of this paper.

Many types of approximators exist, e.g., linearly or nonlinearly parametrized, nonparametric, fixed or adapted while learning, as well as more specialized representations such as factored or relational. Many approximators can be seen as *generalizing* values over certain areas of the state(-action) space. Selecting the right approximator is crucial in successfully solving an RL problem, because the approximator affects the behavior of the algorithm, the quality of the solution obtained, the requirements on the data to use (e.g., its distribution, how to explore in online RL), etc. To reasonably restrict the scope of the paper, we make the following choices:

(i) We focus on parametric approximation, at places considering linear parametrizations in more detail. For instance, a linearly parameterized Q-function approximator uses n basis functions (BFs) $\phi_1, \ldots, \phi_n : X \times U \to \mathbb{R}$ and a parameter vector $\theta \in \mathbb{R}^n$. Approximate Q-values are computed with:

$$\widehat{Q}(x, u; \theta) = \sum_{i=1}^{n} \phi_i(x, u)\theta_i = \phi^{\mathrm{T}}(x, u)\theta \tag{10}$$

where $\phi(x, u) = [\phi_1(x, u), \ldots, \phi_n(x, u)]^{\mathrm{T}}$ is the vector of BFs. An example of nonlinearly parameterized approximator is a multilayer neural network. Due to their simplicity, these predefined approximators are useful to understand the principles of approximate RL. We point out more advanced approximators in Sec. VII.

(ii) We assume that maximal Q-values (and, consequently, greedy actions) can be found efficiently, see, e.g., (6), (8). A widely used approach to guarantee this is to employ only a few discrete actions; in this case, maximal Q-values can be found by enumerating over these actions. Generally, for example when actions are continuous, finding maximal Q-values involves difficult optimization problems that can only be solved approximately.

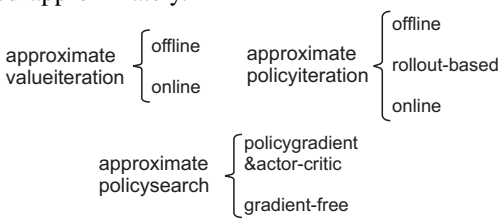

Fig. 2. A taxonomy of approximate RL algorithms.

The upcoming presentation of approximate RL algorithms is organized along the taxonomy shown in Fig. 2. Note that approximate policy search is related to approximate policy iteration; for instance, some policy search algorithms use policy evaluation to estimate the score of a policy, exploiting the fact that $R^h(x) = V^h(x) = Q^h(x, h(x))$. We distinguish the two classes by requiring that policy iteration performs policy improvements that are – with the possible exception of exploration – *fully greedy* (8), whereas policy search improves policies using traditional optimization techniques, such as gradient-based updates.

Due to space limitations and the great breadth of the field, we only introduce a selection of representative algorithms. In making this selection, we prefer established, important methods that have each generated their own subfield of research. We describe these methods in depth, and discuss the convergence and solution quality guarantees of the different classes of algorithms. We thus hope to provide the reader with a good understanding of the structure of the field.

IV. Approximate value iteration

In this section and the next two, we review algorithms for approximate RL. Note that, although X and U were assumed countable, the algorithms introduced are also applicable to uncountable (e.g., continuous) state-action spaces.

A. Offline value iteration

In the offline, model-free case, a batch of transition samples is available:

$$\{(x_j, u_j, x'_j, r_j) \,|\, j = 1, \ldots, N\} \tag{11}$$

where for every j, the next state x'_j and the reward r_j have been observed as a result of taking action u_j in the state x_j. The transition samples may be independent, they may belong to a set of trajectories, or to a single trajectory.

A prominent example of offline, approximate value iteration is fitted Q-iteration [16]–[18]. To understand how this algorithm

is obtained, consider first classical Q-iteration (Sec. II), but in the *deterministic* case. In this case, the dynamics simplify to $x_{k+1} = \overline{f}(x_k, u_k)$, the reward function to $r_{k+1} = \overline{\rho}(x_k, u_k)$, and the Q-iteration update becomes:

$$Q_{\ell+1}(x,u) = \overline{\rho}(x,u) + \gamma \max_{u'} Q_\ell(\overline{f}(x,u), u') \tag{12}$$

To obtain fitted Q-iteration, three changes are made: (i) Since the exact Q-function $Q_\ell(x,u)$ is unavailable, an approximate Q-function $\widehat{Q}(x,u;\theta_\ell)$, parameterized by θ_ℓ, is used instead. (ii) Target Q-values for the updates are only computed for the pairs (x_j, u_j). For each j, the (unknown) $\overline{f}(x_j,u_j)$ and $\overline{\rho}(x_j,u_j)$ are replaced by the observations x'_j and r_j, leading to the target Q-value sample: $q^{\text{targ}}_{\ell+1,j} = r_j + \gamma \max_{u'} \widehat{Q}(x'_j, u'; \theta_\ell)$. Because the system is deterministic, $x'_j = \overline{f}(x_j,u_j)$ and $r_j = \overline{\rho}(x_j,u_j)$; thus, the replacements are exact. (Note also the similarity with the Q-learning update target in (6).) (iii) Rather than computing an entire updated Q-function $Q_{\ell+1}$, *regression* is performed on the available samples, to find a new parameter vector $\theta_{\ell+1}$ so that $\widehat{Q}(x_j,u_j;\theta_{\ell+1}) \approx q_{\ell+1,j}$.

Thus, the entire fitted Q-iteration update can be written:

$$\theta_{\ell+1} = \text{regr}\left\{(x_j,u_j), r_j + \gamma \max_{u'} \widehat{Q}(x'_j, u'; \theta_\ell)\right\} \tag{13}$$

where the generic regression algorithm 'regr' can be instantiated, e.g., to least-squares regression.

In the stochastic case, the target Q-value $Q^{\text{targ}}_{\ell+1}(x_j,u_j)$ is $\mathrm{E}_{x' \sim f(x_j,u_j,\cdot)}\{\rho(x_j,u_j,x') + \gamma \max_{u'} \widehat{Q}(x',u';\theta_\ell)\}$, i.e., an expectation of a random variable, of which $q^{\text{targ}}_{\ell+1,j}$ is *only a sample*. Nevertheless, most regression algorithms, including least-squares, seek to approximate the expected value of their output variable conditioned by the input. This means that the regression actually looks for $\theta_{\ell+1}$ such that $\widehat{Q}(x,u;\theta_{\ell+1}) \approx Q^{\text{targ}}_{\ell+1}(x,u)$, and therefore the update (13) remains valid in the stochastic case.

B. Online value iteration

Online RL algorithms must collect their own data, by interacting with the system while they learn. Similarly to the classical case, approximate Q-learning is the most widely used online algorithm from the approximate value iteration class. Classical Q-learning (6) can be easily combined with parametric approximation by using the gradient-based update (see [1, Ch. 8]):

$$\theta_{k+1} = \theta_k + \alpha_k \big[r_{k+1} + \gamma \max_{u'} \widehat{Q}(x_{k+1}, u'; \theta_k) - \widehat{Q}(x_k, u_k; \theta_k)\big] \frac{\partial \widehat{Q}(x_k,u_k;\theta_k)}{\partial \theta_k} \tag{14}$$

The square brackets contain an approximation of the temporal difference. Approximate Q-learning has been used with a variety of approximators, including linear parametrizations [19], [20], fuzzy rule-bases [21], neural networks [22], etc.

A 'pseudo-online' category of algorithms can be obtained by interspersing offline updates with online episodes in which the currently available solution (e.g., an ε-greedy policy in the current Q-function) is used to collect new, more informative samples [11], [23].

C. Convergence issues

Convergence and near-optimality guarantees for *offline* value iteration often rely on contraction properties [24], [16]. For instance, the Q-iteration update (5) is a contraction. If additionally the approximator $\widehat{Q}(x,u;\theta)$ and the regression algorithm in (13) are nonexpansions, the entire fitted Q-iteration update is a contraction, and therefore asymptotically converges. More recently, so-called 'finite-sample' theoretical results have been developed, which do not rely on contracting updates, but which provide probabilistic near-optimality bounds on the solution obtained after a finite number of iterations, by using a finite number of samples [25], [18], [26].

Convergence of *online* approximate Q-learning was proven for linear parameterizations, under the restrictive requirement of using a fixed policy [19], [20], [27]. Some of these results also require non-expansive approximation [19], [20].

Another theoretical question is consistency: whether the algorithm converges to an optimal solution, asymptotically as the approximation power increases. Some approximate value iteration algorithms are provably consistent [24], [20].

V. Approximate policy iteration

A. Offline policy iteration

In Sec. III, it was assumed that greedy actions can be computed efficiently. This means the *policy improvement* step (8) of approximate policy iteration can be performed exactly and efficiently, on demand for every state where an improved action is necessary. Hence, we mainly focus on the *policy evaluation* problem.

One class of offline policy evaluation can be obtained similarly to offline value iteration, e.g., by performing iterative updates based on the Bellman equation for Q^h (3). Such a fitted policy evaluation algorithm was proposed in [28].

However, if a linear Q-function parametrization (10) is used, $\widehat{Q}(x,u;\theta) = \phi^{\mathrm{T}}(x,u)\theta$, a more specialized class of policy evaluation algorithms can be obtained. Consider the following mapping derived from the Bellman equation (3):

$$[T^h(Q)](x,u) = \mathrm{E}_{x' \sim f(x,u,\cdot)}\{\rho(x,u,x') + \gamma Q(x', h(x'))\} \tag{15}$$

Classical policy evaluation aims to solve the Bellman equation $Q^h = T^h(Q^h)$, which is equivalent to (3). *Projected policy evaluation* algorithms instead look for a solution of the *projected Bellman equation* [6, Ch. 6]:[5]

$$\widehat{Q}^h = P^w(T^h(\widehat{Q}^h)) \tag{16}$$

where the mapping P^w performs a weighted least-squares projection onto the space of representable Q-functions, i.e., on $\{\phi^{\mathrm{T}}(x,u)\theta \mid \theta \in \mathbb{R}^n\}$. The weight function $w : X \times U \to [0,1]$ controls the distribution of the approximation error, and is also used as a probability distribution, so it must satisfy $\sum_{x,u} w(x,u) = 1$. To guarantee that $P^w(T^h)$ is a contraction

[5] Another important class of policy evaluation approaches aims to minimize the *Bellman error* (residual), which is the difference between the two sides of the Bellman equation: $\int_{X\times U} (\widehat{Q}^h(x,u) - [T^h(\widehat{Q}^h)](x,u))^2 \mathrm{d}(x,u)$, see, e.g., [29], [30].

and (16) has a unique solution, the weight of each state-action pair should be the steady-state probability of this pair occurring along an infinitely-long trajectory generated with h [6, Sec. 6.3]. In practice, however, (16) also has meaningful solutions for other weight functions [31].

By exploiting the linearity of the approximator in the parameters in combination with the linearity of the mapping T^h in the Q-values, (16) can eventually be written as a linear equation in the parameter vector: $\Gamma\theta^h = z$, where $\Gamma \in \mathbb{R}^{n\times n}$ and $z \in \mathbb{R}^n$ can be estimated from transition samples. Consider a batch of N samples (11), so that the sampling probability of each pair (x, u) is $w(x, u)$. The estimates of Γ and z are initialized to zeros and updated with:

$$\begin{aligned} \Gamma_j &= \Gamma_{j-1} + \phi(x_j, u_j)\left[\phi(x_j, u_j) - \gamma\phi(x'_j, h(x'_j))\right]^{\mathrm{T}} \\ z_j &= z_{j-1} + \phi(x_j, u_j)r_j \end{aligned} \tag{17}$$

Least-squares temporal difference for Q-functions (LSTD-Q) [31], [32] is a policy evaluation algorithm that processes all the samples using (17) and then solves:

$$(\Gamma_N/N)\widehat{\theta}^h = z_N/N$$

to find an approximate parameter vector $\widehat{\theta}^h$. When $N \to \infty$, $\Gamma_N/N \to \Gamma$ and $z_N/N \to z$, so that $\widehat{\theta}^h \to \theta^h$. By combining LSTD-Q policy evaluation with exact policy improvements (8), an offline algorithm for approximate policy iteration is obtained, called least-squares policy iteration [31], [33]. An alternative to LSTD-Q is the so-called least-squares policy evaluation for Q-functions (LSPE-Q) [34], [35].

B. Rollout-based policy evaluation

Rollout-based algorithms for policy evaluation do not represent the value function explicitly, but evaluate it on demand, using Monte Carlo simulations. For example, to estimate the Q-value $\widehat{Q}^h(x, u)$ of a given state-action pair (x, u), a number N_{MC} of trajectories are simulated, where each trajectory has length K and starts from the pair (x, u). The estimated Q-value is then the average of the sample returns obtained along these trajectories:

$$\widehat{Q}^h(x, u) = \frac{1}{N_{\mathrm{MC}}}\sum_{i_0=1}^{N_{\mathrm{MC}}}\left[\rho(x, u, x_{i_0,1}) + \sum_{k=1}^{K}\gamma^k\rho(x_{i_0,k}, h(x_{i_0,k}), x_{i_0,k+1})\right]$$

For each trajectory i_0, the first state-action pair is fixed to (x, u) and leads to $x_{i_0,1} \sim f(x, u, \cdot)$. Thereafter, actions are chosen using the policy h, thus $x_{i_0,k+1} \sim f(x_{i_0,k}, h(x_{i_0,k}), \cdot)$. This procedure is called a *rollout* [36], [37].

Using rollouts can be computationally expensive, especially in the stochastic case, and the computational cost is proportional to the number of points at which the value function must be evaluated. Therefore, rollouts are most useful when this number is small, and Bellman-equation based methods may be preferable when the value function must be evaluated at many (or all) points.

C. Online policy iteration

So-called *optimistic* policy iteration algorithms can be obtained from offline algorithms, by collecting samples online and performing policy improvements once every few samples, without waiting to complete an accurate evaluation of the current policy. Optimistic policy improvements have been combined with, e.g., LSPE-Q [38] and LSTD-Q [39]. Approximate SARSA [40], an algorithm obtained from SARSA by using gradient-based updates, can also be seen as a kind of optimistic policy iteration. It updates the parameters with:

$$\begin{aligned} \theta_{k+1} = \theta_k + \alpha_k\big[&r_{k+1} + \gamma\widehat{Q}(x_{k+1}, u_{k+1}; \theta_k) \\ &- \widehat{Q}(x_k, u_k; \theta_k)\big]\frac{\partial}{\partial\theta_k}\widehat{Q}(x_k, u_k; \theta_k) \end{aligned} \tag{18}$$

similarly to approximate Q-learning (14). Actions should be chosen with an exploratory policy derived from a policy greedy in $\widehat{Q}(x, u; \theta_k)$.

Offline policy iteration can be used in a 'pseudo-online' way (as explained in Sec. IV-B), by making offline updates in-between online episodes of samples collection [41]. Extensions of gradient-based policy evaluation are given in [42], [43].

D. Convergence issues

If the policy evaluation and policy improvement errors are bounded at each iteration, offline policy iteration eventually produces policies with a bounded suboptimality, [31], [4, Sec. 6.2]. (In the algorithms above, only exact policy improvements were considered, leading to a zero policy improvement error.) However, in general the convergence to a fixed policy is not guaranteed; instead, the policy may, e.g., oscillate. Finite-sample results are also available [29], [30], [44]. Many theoretical works focus on policy evaluation only [45]–[47].

Under appropriate conditions, approximate SARSA converges to a fixed point [27]. The behavior of general optimistic policy iteration has not been properly understood yet, and can be very complicated. For instance, a phenomenon called chattering can occur, whereby the value function converges while the policy sequence oscillates [4, Sec. 6.4].

VI. Approximate policy search

Algorithms for approximate policy search represent the policy explicitly, usually by a parametric approximator, and seek an optimal parameter vector using optimization techniques.

A. Policy gradient and actor-critic algorithms

In policy gradient methods, the policy is represented using a differentiable parametrization, and gradient updates are performed to find parameters that lead to (locally) maximal returns [8]. The policy is usually stochastic, to include exploration: $u \sim h(x, u; \vartheta)$, where $\vartheta \in \mathbb{R}^{\mathcal{N}}$ is the policy parameter.

Policy gradient methods are usually formalized under the average return criterion, different from the discounted return (1) considered in the other sections. The expected average return from x_0 under the policy parameterized by ϑ is:

$$R^\vartheta(x_0) = \lim_{K\to\infty}\frac{1}{K}\mathrm{E}_{\substack{u_k\sim h(x_k,\cdot;\vartheta),\\ x_{k+1}\sim f(x_k,u_k,\cdot)}}\left\{\sum_{k=0}^{K}\rho(x_k, u_k, x_{k+1})\right\}$$

Under appropriate conditions, see [6, Ch. 4], the average return is the same for every initial state (denote it by $\mathcal{R}^\vartheta$), and together with a so-called differential V-function, satisfies a variant of Bellman equation.

A parameter ϑ^* that (locally) maximizes $\mathcal{R}^\vartheta$ must be found. To this end, gradient ascent updates are performed:

$$\vartheta \leftarrow \vartheta + \alpha \frac{\partial \mathcal{R}^\vartheta}{\partial \vartheta} \tag{19}$$

where α is the step size. The core problem is estimating $\frac{\partial \mathcal{R}^\vartheta}{\partial \vartheta}$.

The gradient can be rewritten in a form that allows it to be estimated using simulations similar to rollouts [48]. An alternative is to explicitly approximate the V-function, leading to actor-critic algorithms (the original actor-critic was given in [49]). The V-function can be found, e.g., by using average-return variants of LSTD and LSPE [6, Sec. 6.6].

Another class of algorithms can be obtained by expressing the gradient in terms of a Q-function. It has been shown [50], [51] that such a gradient estimate remains valid even if the Q-function is linearly approximated, given that a set of so-called *compatible* BFs, derived from the policy parametrization, are used. Actor-critic variants that exploit such a parameterization have been given, e.g., in [50], [51].

The so-called 'natural' policy gradient [52] leads to faster convergence than the 'vanilla' policy gradient in (19), besides other advantages [8]. Natural policy gradients were used to develop natural actor-critic algorithms in, e.g., [53], [54].

Policy gradient and actor-critic algorithms provably converge to a local optimum under mild conditions.

B. Gradient-free policy search

Gradient-based policy search is based on the assumptions that the policy is differentiable and that the locally optimal parameters found by the gradient method are near the global optimum. When these assumptions are not satisfied, global, gradient-free optimization algorithms must be used to search for the policy parameters. For instance, the problem may require a rich policy parametrization that is not differentiable, or that leads to a return landscape with many local optima.

Many gradient-free optimization techniques can be used, including evolutionary computation, tabu search, pattern search, the cross-entropy method, etc. For instance, policy search was combined with evolutionary computation in [55], [56, Ch. 3], and with cross-entropy optimization in [57], [58].

VII. Discussion and outlook

Comparison of approaches: A general, qualitative comparison of the classes of algorithms introduced is provided next. In the offline case, policy iteration often converges in a small number of iterations [1, Sec. 4.3], possibly smaller than value iteration. However, this does not mean that policy iteration is computationally less demanding, since policy evaluation is a difficult problem in itself, and must be solved at every policy iteration. In certain conditions, approximate value iteration provably converges to a *unique* solution [59], whereas this is more difficult to guarantee for approximate policy iteration, which generally only converges to a *sequence* of policies that all provide a guaranteed performance [4].

Offline algorithms like fitted Q-iteration and least-squares policy iteration typically exploit data more efficiently than gradient-based, online algorithms such as approximate Q-learning and approximate SARSA; this comes at the cost of increased computational complexity. Interspersing offline updates with online sample collection leads to 'pseudo-online', yet still data-efficient, algorithms (see the 'Outlook' below for other ways to improve data efficiency).

Within the approximate policy search class, policy gradient and actor-critic algorithms have useful convergence guarantees and moderate computational demands. They can be applied when an appropriate, differentiable policy parametrization can be found, which makes them very suitable to certain classes of problems such as robot control [8]. However, their convergence point is only *locally* optimal, which is a disadvantage with respect to value and policy iteration, which do provide *global* performance guarantees.

Even when a good policy parametrization cannot be obtained, approximate policy search can still be useful in its gradient-free, global optimization forms, which can however incur larger computational costs. An important benefit of policy search is that, in contrast to value and policy iteration, it can handle continuous action spaces without difficulty.

Outlook: We outline some important issues in approximate RL that, due to space limitations, could not be addressed in detail in the survey.

Designing a good parametric approximator for the value function is difficult; *nonparametric* and *adaptive* approximators can alleviate this problem (potentially at the cost of endangering convergence). Such approximators have been extensively studied, e.g., for value iteration [24], [16], [26] (nonparametric), [20], [60]–[62] (adaptive), as well as for policy evaluation and iteration [63], [38], [30], [64] (nonparametric), [65]–[68] (adaptive).

For value and policy iteration, it was assumed that Q-function maximizations can be performed easily. This is not true in general, especially when continuous actions are needed. Methods that deal with continuous actions were given in, e.g., [69], [18], [23], [70].

The data efficiency of online methods can be increased by: using eligibility traces, [1, Ch. 7]; reusing stored transition samples, in the *experience replay* approaches [22]; and building a model, which can then be used to generate new samples, in the so-called Dyna or *model-learning* methods [71], [9].

Other important issues include RL in relational domains [72], efficient exploration [73], [74], exploiting problem decompositions to scale RL methods [75], solving tasks in which not all the state variables can be perceived (called 'partially observable'), approximate RL for multiple agents, etc.

Further reading: We have largely focused on the RL literature from the AI field, and did not cover the rich, control-oriented field called 'approximate DP', which despite its name does provide RL algorithms. Good overviews of this field are

given in the edited book [5] and the special issue [76].

Extensive accounts of RL and DP can be found in the books: [1]–[3] (artificial-intelligence oriented), [4], [6] (optimal-control oriented), [77] (operations-research oriented), [78] (perturbation-theory oriented), [56] (with a focus on simulation-based methods), as well as our recent book [7], which focuses explicitly on approximate RL and DP.

The recent overview texts [79], [35] center on policy iteration and least-squares policy evaluation, while touching many other topics; [80] outlines RL from a machine learning perspective. For surveys of DP, see [81] (control-theoretic perspective) and [82] (economics perspective).

REFERENCES

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

[2] Cs. Szepesvári, *Algorithms for Reinforcement Learning*. Morgan & Claypool Publishers, 2010.

[3] O. Sigaud and O. Buffet, Eds., *Markov Decision Processes in Artificial Intelligence*. Wiley, 2010.

[4] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[5] J. Si, A. G. Barto, and W. B. Powell, Eds., *Handbook of Learning and Approximate Dynamic Programming*. Wiley, 2004.

[6] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 3rd ed. Athena Scientific, 2007, vol. 2.

[7] L. Buşoniu, R. Babuška, B. De Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*, ser. Automation and Control Engineering. Taylor & Francis CRC Press, 2010.

[8] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural Networks*, vol. 21, pp. 682–697, 2008.

[9] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," in *Advances in Neural Information Processing Systems 19*, B. Schölkopf, J. C. Platt, and T. Hoffman, Eds. MIT Press, 2007, pp. 1–8.

[10] J. DiGiovanna, B. Mahmoudi, J. Fortes, J. C. Principe, and J. C. Sanchez, "Coadaptive brain-machine interface via reinforcement learning," *IEEE Transactions on Biomedical Engineering*, vol. 56, no. 1, pp. 54–64, 2009.

[11] D. Ernst, G.-B. Stan, J. Gonçalves, and L. Wehenkel, "Clinical data based optimal STI strategies for HIV: A reinforcement learning approach," in *Proceedings 45th IEEE Conference on Decision & Control*, San Diego, US, 13–15 December 2006, pp. 667–672.

[12] D. P. Bertsekas and S. E. Shreve, *Stochastic Optimal Control: The Discrete Time Case*. Academic Press, 1978.

[13] 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.

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

[15] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," Engineering Department, Cambridge University, UK, Tech. Rep. CUED/F-INFENG/TR166, September 1994, available at http://mi.eng.cam.ac.uk/reports/svr-ftp/rummery_tr166.ps.Z.

[16] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *Journal of Machine Learning Research*, vol. 6, pp. 503–556, 2005.

[17] M. Riedmiller, "Neural fitted Q-iteration – first experiences with a data efficient neural reinforcement learning method," in *Proceedings 16th European Conference on Machine Learning (ECML-05)*, ser. Lecture Notes in Computer Science, vol. 3720, Porto, Portugal, 3-7 October 2005, pp. 317–328.

[18] A. Antos, R. Munos, and Cs. Szepesvári, "Fitted Q-iteration in continuous action-space MDPs," in *Advances in Neural Information Processing Systems 20*, J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, Eds. MIT Press, 2008, pp. 9–16.

[19] S. P. Singh, T. Jaakkola, and M. I. Jordan, "Reinforcement learning with soft state aggregation," in *Advances in Neural Information Processing Systems 7*, G. Tesauro, D. S. Touretzky, and T. K. Leen, Eds. MIT Press, 1995, pp. 361–368.

[20] Cs. Szepesvári and W. D. Smart, "Interpolation-based Q-learning," in *Proceedings 21st International Conference on Machine Learning (ICML-04)*, Bannf, Canada, 4–8 July 2004, pp. 791–798.

[21] P. Y. Glorennec, "Reinforcement learning: An overview," in *Proceedings European Symposium on Intelligent Techniques (ESIT-00)*, Aachen, Germany, 14–15 September 2000, pp. 17–35.

[22] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, no. 3–4, pp. 293–321, Aug. 1992, special issue on reinforcement learning.

[23] G. Neumann and J. Peters, "Fitted Q-iteration by advantage weighted regression," in *Advances in Neural Information Processing Systems 21*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds. MIT Press, 2009, pp. 1177–1184.

[24] D. Ormoneit and S. Sen, "Kernel-based reinforcement learning," *Machine Learning*, vol. 49, no. 2–3, pp. 161–178, 2002.

[25] R. Munos and Cs. Szepesvári, "Finite time bounds for fitted value iteration," *Journal of Machine Learning Research*, vol. 9, pp. 815–857, 2008.

[26] A. M. Farahmand, M. Ghavamzadeh, Cs. Szepesvári, and S. Mannor, "Regularized fitted Q-iteration for planning in continuous-space Markovian decision problems," in *Proceedings 2009 American Control Conference (ACC-09)*, St. Louis, US, 10–12 June 2009, pp. 725–730.

[27] F. S. Melo, S. P. Meyn, and M. I. Ribeiro, "An analysis of reinforcement learning with function approximation," in *Proceedings 25th International Conference on Machine Learning (ICML-08)*, Helsinki, Finland., 5–9 July 2008, pp. 664–671.

[28] S. Jodogne, C. Briquet, and J. H. Piater, "Approximate policy iteration for closed-loop learning of visual tasks," in *Proceedings 17th European Conference on Machine Learning (ECML-06)*, ser. Lecture Notes in Computer Science, vol. 4212, Berlin, Germany, 18–22 September 2006, pp. 210–221.

[29] A. Antos, Cs. Szepesvári, and R. Munos, "Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path," *Machine Learning*, vol. 71, no. 1, pp. 89–129, 2008.

[30] A. M. Farahmand, M. Ghavamzadeh, Cs. Szepesvári, and S. Mannor, "Regularized policy iteration," in *Advances in Neural Information Processing Systems 21*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds. MIT Press, 2009, pp. 441–448.

[31] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.

[32] S. J. Bradtke and A. G. Barto, "Linear least-squares algorithms for temporal difference learning," *Machine Learning*, vol. 22, no. 1–3, pp. 33–57, 1996.

[33] C. Thiery and B. Scherrer, "Least-squares λ policy iteration: Bias-variance trade-off in control problems," in *Proceedings 27th International Conference on Machine Learning (ICML-10)*, Haifa, Israel, 21–24 June 2010, pp. 1071–1078.

[34] D. P. Bertsekas and S. Ioffe, "Temporal differences-based policy iteration and applications in neuro-dynamic programming," Massachusetts Institute of Technology, Cambridge, US, Tech. Rep. LIDS-P-2349, 1996, available at http://web.mit.edu/dimitrib/www/Tempdif.pdf.

[35] D. P. Bertsekas, "Approximate policy iteration: A survey and some new methods," Massachusetts Institute of Technology, Cambridge, US, Tech. Rep. LIDS 2833, July 2010.

[36] M. G. Lagoudakis and R. Parr, "Reinforcement learning as classification: Leveraging modern classifiers," in *Proceedings 20th International Conference on Machine Learning (ICML-03)*. Washington, US, 21–24 August 2003, pp. 424–431.

[37] C. Dimitrakakis and M. Lagoudakis, "Rollout sampling approximate policy iteration," *Machine Learning*, vol. 72, no. 3, pp. 157–171, 2008.

[38] T. Jung and D. Polani, "Learning RoboCup-keepaway with kernels," in *Gaussian Processes in Practice*, ser. JMLR Workshop and Conference Proceedings, vol. 1, 2007, pp. 33–57.

[39] L. Buşoniu, D. Ernst, B. De Schutter, and R. Babuška, "Online least-squares policy iteration for reinforcement learning control," in *Proceedings 2010 American Control Conference (ACC-10)*, Baltimore, US, 30 June – 2 July 2010, pp. 486–491.

[40] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Advances in Neural Information Processing Systems 8*, D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, Eds. MIT Press, 1996, pp. 1038–1044.

[41] L. Li, M. L. Littman, and C. R. Mansley, "Online exploration in least-squares policy iteration," in *Proceedings 8th International Joint*

Conference on Autonomous Agents and Multiagent Systems (AAMAS-09), vol. 2, Budapest, Hungary, 10–15 May 2009, pp. 733–739.

[42] R. Sutton, H. Maei, D. Precup, S. Bhatnagar, D. Silver, Cs. Szepesvari, and E. Wiewiora, "Fast gradient-descent methods for temporal-difference learning with linear function approximation," in *Proceedings 26th International Conference on Machine Learning (ICML-09)*, Montreal, Canada, 14–18 June 2009, pp. 993–1000.

[43] H. R. Maei, C. Szepesvári, S. Bhatnagar, and R. S. Sutton, "Toward off-policy learning control with function approximation," in *Proceedings 27th International Conference on Machine Learning (ICML-10)*, Haifa, Israel, 21–24 June 2010, pp. 719–726.

[44] A. Lazaric, M. Ghavamzadeh, and R. Munos, "Analysis of a classification-based policy iteration algorithm," in *Proceedings 27th International Conference on Machine Learning (ICML-10)*, Haifa, Israel, 21–24 June 2010, pp. 607–614.

[45] H. Yu, "Convergence of least squares temporal difference methods under general conditions," in *Proceedings 27th International Conference on Machine Learning (ICML-10)*, Haifa, Israel, 21–24 June 2010, pp. 1207–1214.

[46] B. Scherrer, "Should one compute the Temporal Difference fix point or minimize the Bellman Residual? the unified oblique projection view," in *Proceedings 27th International Conference on Machine Learning (ICML-10)*, Haifa, Israel, 21–24 June 2010, pp. 959–966.

[47] A. Lazaric, M. Ghavamzadeh, and R. Munos, "Finite-sample analysis of LSTD," in *Proceedings 27th International Conference on Machine Learning (ICML-10)*, Haifa, Israel, 21–24 June 2010, pp. 615–622.

[48] P. Marbach and J. N. Tsitsiklis, "Approximate gradient methods in policy-space optimization of Markov reward processes," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 13, no. 1–2, pp. 111–148, 2003.

[49] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 13, no. 5, pp. 833–846, 1983.

[50] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems 12*, S. A. Solla, T. K. Leen, and K.-R. Müller, Eds. MIT Press, 2000, pp. 1057–1063.

[51] V. R. Konda and J. N. Tsitsiklis, "On actor-critic algorithms," *SIAM Journal on Control and Optimization*, vol. 42, no. 4, pp. 1143–1166, 2003.

[52] S. Kakade, "A natural policy gradient," in *Advances in Neural Information Processing Systems 14*, T. G. Dietterich, S. Becker, and Z. Ghahramani, Eds. MIT Press, 2001, pp. 1531–1538.

[53] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7–9, pp. 1180–1190, 2008.

[54] S. Bhatnagar, R. Sutton, M. Ghavamzadeh, and M. Lee, "Natural actor-critic algorithms," *Automatica*, vol. 45, no. 11, pp. 2471–2482, 2009.

[55] F. J. Gomez, J. Schmidhuber, and R. Miikkulainen, "Efficient non-linear control through neuroevolution," in *Proceedings 17th European Conference on Machine Learning (ECML-06)*, ser. Lecture Notes in Computer Science, vol. 4212, Berlin, Germany, 18-22 September 2006, pp. 654–662.

[56] H. S. Chang, M. C. Fu, J. Hu, and S. I. Marcus, *Simulation-Based Algorithms for Markov Decision Processes*. Springer, 2007.

[57] S. Mannor, R. Y. Rubinstein, and Y. Gat, "The cross-entropy method for fast policy search," in *Proceedings 20th International Conference on Machine Learning (ICML-03)*, Washington, US, 21–24 August 2003, pp. 512–519.

[58] L. Buşoniu, D. Ernst, B. De Schutter, and R. Babuška, "Cross-entropy optimization of control policies with adaptive basis functions," *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, vol. 41, no. 1, 2011, accepted for publication, available online.

[59] J. N. Tsitsiklis and B. Van Roy, "Feature-based methods for large scale dynamic programming," *Machine Learning*, vol. 22, no. 1–3, pp. 59–94, 1996.

[60] S. Whiteson and P. Stone, "Evolutionary function approximation for reinforcement learning," *Journal of Machine Learning Research*, vol. 7, pp. 877–917, 2006.

[61] P. W. Keller, S. Mannor, and D. Precup, "Automatic basis function construction for approximate dynamic programming and reinforcement learning," in *Proceedings 23rd International Conference on Machine Learning (ICML-06)*, Pittsburgh, US, 25–29 June 2006, pp. 449–456.

[62] P. Preux, S. Girgin, and M. Loth, "Feature discovery in approximate dynamic programming," in *Proceedings IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL-09)*, 30 March – 2 April 2009, pp. 109–116.

[63] Y. Engel, S. Mannor, and R. Meir, "Reinforcement learning with Gaussian processes," in *Proceedings 22nd International Conference on Machine Learning (ICML-05)*, Bonn, Germany, 7–11 August 2005, pp. 201–208.

[64] G. Taylor and R. Parr, "Kernelized value function approximation for reinforcement learning," in *Proceedings 26th International Conference on Machine Learning (ICML-09)*, Montreal, Canada, 14–18 June 2009, pp. 1017–1024.

[65] I. Menache, S. Mannor, and N. Shimkin, "Basis function adaptation in temporal difference reinforcement learning," *Annals of Operations Research*, vol. 134, no. 1, pp. 215–238, 2005.

[66] S. Mahadevan and M. Maggioni, "Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes," *Journal of Machine Learning Research*, vol. 8, pp. 2169–2231, 2007.

[67] J. Z. Kolter and A. Ng, "Regularization and feature selection in least-squares temporal difference learning," in *Proceedings 26th International Conference on Machine Learning (ICML-09)*, Montreal, Canada, 14–18 June 2009, pp. 521–528.

[68] R. Parr, L. Li, G. Taylor, C. Painter-Wakefield, and M. Littman, "An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning," in *Proceedings 25th Annual International Conference on Machine Learning (ICML-08)*, Helsinki, Finland, 5–9 July 2008, pp. 752–759.

[69] H. van Hasselt and M. Wiering, "Reinforcement learning in continuous action spaces," in *Proceedings IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL-07)*, Honolulu, US, 1–5 April 2007, pp. 272–279.

[70] J. Pazis and M. Lagoudakis, "Binary action search for learning continuous-action control policies," in *Proceedings of the 26th International Conference on Machine Learning (ICML-09)*, Montreal, Canada, 14–18 June 2009, pp. 793–800.

[71] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *Proceedings 7th International Conference on Machine Learning (ICML-90)*, Austin, US, 21–23 June 1990, pp. 216–224.

[72] M. van Otterlo, *The Logic Of Adaptive Behavior*. IOS Press, 2009.

[73] J.-Y. Audibert, R. Munos, and Cs. Szepesvári, "Tuning bandit algorithms in stochastic environments," in *Proceedings 18th International Conference on Algorithmic Learning Theory (ALT-07)*, Sendai, Japan, 1–4 October 2007, pp. 150–165.

[74] S. Bubeck, R. Munos, G. Stoltz, and C. Szepesvári, "Online optimization in X-armed bandits," in *Advances in Neural Information Processing Systems 21*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds. MIT Press, 2009, pp. 201–208.

[75] B. Kveton, M. Hauskrecht, and C. Guestrin, "Solving factored MDPs with hybrid state and action variables," *Journal of Artificial Intelligence Research*, vol. 27, pp. 153–201, 2006.

[76] F. Lewis, D. Liu, and G. Lendaris, "Guest editorial - special issue on adaptive dynamic programming and reinforcement learning in feedback control," *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, vol. 38, no. 4, pp. 896–897, 2008.

[77] W. B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Wiley, 2007.

[78] X.-R. Cao, *Stochastic Learning and Optimization: A Sensitivity-Based Approach*. Springer, 2007.

[79] D. P. Bertsekas, "Approximate dynamic programming," 20 November 2010, update of Chapter 6 in volume 2 of the book Dynamic Programming and Optimal Control. Available at http://web.mit.edu/dimitrib/www/dpchapter.html.

[80] V. Heidrich-Meisner, M. Lauer, C. Igel, and M. Riedmiller, "Reinforcement learning in a nutshell," in *Proceedings 15th European Symposium on Artificial Neural Networks (ESANN-07)*, Bruges, Belgium, 25–27 April 2007, pp. 277–288.

[81] D. P. Bertsekas, "Dynamic programming and suboptimal control: A survey from ADP to MPC," *European Journal of Control*, vol. 11, no. 4–5, pp. 310–334, 2005, special issue for the CDC-ECC-05 in Seville, Spain.

[82] J. Rust, "Numerical dynamic programming in economics," in *Handbook of Computational Economics*, H. M. Amman, D. A. Kendrick, and J. Rust, Eds. Elsevier, 1996, vol. 1, ch. 14, pp. 619–729.