Weighted Bellman Equations and their Applications in Approximate Dynamic Programming*

Huizhen Yu† Dimitri P. Bertsekas‡

Abstract

We consider approximation methods for Markov decision processes in the learning and simulation context. For policy evaluation based on solving approximate versions of a Bellman equation, we propose the use of weighted Bellman mappings. Such mappings comprise weighted sums of one-step and multistep Bellman mappings, where the weights depend on both the step and the state. For projected versions of the associated Bellman equations, we show that their solutions have the same nature and essential approximation properties as the commonly used approximate solutions from TD(λ).

The most important feature of our framework is that each state can be associated with a different type of mapping. Compared with the standard TD(λ) framework, this gives a more flexible way to combine multistage costs and state transition probabilities in approximate policy evaluation, and provides alternative means for bias-variance control. With weighted Bellman mappings, there is also greater flexibility to design learning and simulation-based algorithms. We demonstrate this with examples, including new TD-type algorithms with state-dependent λ parameters, as well as block versions of the algorithms. Weighted Bellman mappings can also be applied in approximate policy iteration: we provide several examples, including some new optimistic policy iteration schemes.

Another major feature of our framework is that the projection need not be based on a norm, but rather can use a semi-norm. This allows us to establish a close connection between projected equation and aggregation methods, and to develop for the first time multistep aggregation methods, including some of the TD(λ)-type.

*Work supported by the Air Force Grant FA9550-10-1-0412.

†Lab. for Information and Decision Systems, M.I.T. janey_yu@mit.edu

‡Lab. for Information and Decision Systems, M.I.T. dimitrib@mit.edu

Contents

1 Overview

Approximate policy evaluation (APE) and approximate policy iteration (API) are two important classes of approximation methods for solving large-scale Markov decision processes (MDP). In this paper we consider APE and API algorithms that are based on reinforcement learning or simulation and require no knowledge of the MDP model [BT96, SB98, Gos03, CFHM07, Pow07, Mey08, BBSE10, Sze10, Ber12]. We will first develop our ideas for the case of APE and then apply and extend them to API.

One approach to APE is to solve low-dimensional approximate versions of Bellman equations. Examples include the widely used state aggregation methods and temporal difference (TD) methods (see e.g., [Gor95, TV96] and [Sut88, BT96, BB96, TV97, Boy99]). Intrinsic to this approach are approximation biases, namely deviations from the best possible approximate solution. They may be reduced by taking into account multistage costs of a policy. In algorithms of TD(λ)-type, this is elegantly done by considering multistep Bellman mappings with weights that decrease geometrically with the number of steps, at a rate λ. Of concern in APE is also estimation variance: it usually increases when using multistage costs, especially in policy evaluation (PE) with exploration [SB98, PSD01]. In TD methods, the bias-variance tradeoff can be controlled by the value of λ (see e.g., [KS00]).

In this paper we propose the use of generally weighted Bellman mappings – with state and step-dependent weights – for cost approximation in MDP. The advantages of such mappings are multifold. Theoretically, as will be discussed later, we already have some understanding of their associated approximate Bellman equations formed by projection: their solutions have the same nature and essential properties as those produced by TD methods. Computationally, we gain considerable flexibility in designing new APE algorithms for bias-variance control: we can choose different types of mappings according to the properties of each state (instead of tuning a single parameter λ as in TD methods). These algorithms also need not be determined by a prescribed Bellman mapping as in TD(λ). Instead, they can be designed according to certain algorithmic guidelines, and the weighted mapping will then be determined by the interplay between these guidelines and the problem at hand. We may not know exactly the weights in the mapping. Nevertheless, the approximate solution produced can still be understood in view of the general properties of the class of all such mappings. An example of such a situation arises in simulation-based computation: the weights are determined by the simulation process, which may in turn be modified adaptively based on the results that it produces.

We describe our approach for finite-space discounted MDP for mathematical simplicity. The ideas are applicable in general-space MDP with discounted or undiscounted cost criteria, and with nonlinear function approximation [MSB$^+$09]. They can also be applied in the context of API with changing policies: for instance, the standard framework with a policy update after policy evaluation, optimistic API (e.g., [TS10, Ber11]), and actor-critic schemes (e.g., [KT03]). We will address some of these applications in the paper.

Regarding the connection with earlier works, general forms of Bellman mappings have appeared in generalized TD algorithms for the lookup table case [Sut95, BT96, SB98] and played an important role in their analyses. These mappings, however, have not been sufficiently studied and systematically applied in the context of approximation methods, to our knowledge. There are only a few related recent works. The paper [MS10] considered the generalized TD algorithm with variable λ [Sut95, SB98] and provided a two-time-scale gradient-based implementation of the algorithm with function approximation. Another recent work, which has influenced our research, is Ueno et al. [UMKI11]. They called attention to a broad class of equations of which the desired cost function is a solution, and by viewing the cost function as a model parameter, they based their work on the statistical framework of estimating equations. In their framework, the associated mappings need not even be Bellman mappings. However, their framework does not address the case of a misspecified

model and how it affects approximation error. Their research emphasis thus differs entirely from ours: they focused on deriving algorithms with minimal asymptotic variance, assuming that the cost function lies in the approximation space, whereas we will focus on projected versions of Bellman equations, address approximation error properties, and explore the diversity of approximation algorithms that can be obtained in the light of general Bellman mappings.

We note, however, that the weighted Bellman mappings we consider in this paper are only a subclass of generalized Bellman mappings, which allow for even a greater variety of approximation algorithms to be designed. It is beyond our scope to investigate the most general class of such mappings; this is deferred to a following paper. However, the approximation error analysis we will give applies to such mappings as well.

Regarding the main contributions of this paper, they include, besides the proposal of the general weighting of Bellman mappings, several other novel results:

- A new feature of our projected Bellman equation framework is that it allows the use of semi-norm projection (Section 2.1). Among others, this characteristic is critical for establishing a close connection between the projected equation and the aggregation approaches for APE. As a result we have obtained, for the first time, multistep versions of aggregation methods, which parallel the TD(λ) methods and have similarly improved approximation error bias characteristics (Section 4).
- We introduce new TD-type algorithms with state-dependent λ parameters, as well as block versions of the algorithms, based on projected weighted Bellman equations (Sections 3.3 and 3.4). They can be implemented efficiently, and they are useful for bias-variance control especially in exploration enhanced PE, and associated optimistic API schemes.
- We propose a modified policy iteration-based API algorithm, which is closer to policy iteration than similar optimistic API algorithms [TS10, Ber11], thereby extending these recent works (Section 5.1.2).

We organize the paper as follows. In Section 2, we describe our main ideas of weighted Bellman equations with semi-norm projection, and discuss their approximation properties and computational aspects. In Section 3, we give several example algorithms for the learning and simulation contexts, including new TD-type algorithms. In Section 4, we relate a class of multistep aggregation methods to projected Bellman equations with semi-norm projection. In Section 5, we present some applications in API.

2 Main Ideas

We consider an MDP on the state space $\{1, 2, \ldots, n\}$ and with discount factor $\beta \in [0, 1)$. Let μ be a given stationary policy, deterministic or randomized. The cost vector of μ, denoted by $x^* = (x_1^*, \ldots, x_n^*)$, is the unique solution of the Bellman equation $x = Tx$, with T being the Bellman mapping

$$Tx = g + \beta P x, \quad x \in \Re^n.$$

Here g is the vector of expected one-stage costs and P is the transition probability matrix of the Markov chain induced by μ. The mapping T is a contraction with modulus β and with respect to the sup-norm (see e.g., [Put94, BT96]).

2.1 Weighted Bellman Equations with Semi-Norm Projection

The focus of this paper will be on a class of weighted Bellman equations. For each state i, let

$$\{c_{ik} \mid k = 1, 2, \ldots, +\infty\}$$

be a scalar sequence such that

$$c_{ik} \geq 0, \qquad \sum_{k\geq 1} c_{ik} = 1. \tag{2.1}$$

Denoting by $\mathbf{c}$ the collection of the n sequences, $\{c_{ik}\}, i = 1, \ldots, n$, we define a mapping $T^{(\mathbf{c})} : \Re^n \mapsto \Re^n$ by

$$T_i^{(\mathbf{c})} = \sum_{k\geq 1} c_{ik} \cdot T_i^k, \qquad i = 1, \ldots, n, \tag{2.2}$$

where $T_i^{(\mathbf{c})}$ and T_i^k denote the ith component of $T^{(\mathbf{c})}$ and T^k respectively, with T^∞ given by

$$T^\infty x \equiv x^*, \qquad \forall\, x.$$

For all $\mathbf{c}$ satisfying (2.1), $T^{(\mathbf{c})}$ is well-defined due to the contraction property of T; $T^{(\mathbf{c})}$ is also a sup-norm contraction with modulus no greater than β, and x^* is the unique solution of

$$x = T^{(\mathbf{c})} x. \tag{2.3}$$

Familiar examples of $T^{(\mathbf{c})}$ include the multistep Bellman mappings T^m, $m \geq 1$, and the λ-weighted mappings associated with the TD(λ)-type methods:

$$\text{for } \lambda \in [0,1), \quad T^{(\lambda)} = (1-\lambda)\sum_{k\geq 1} \lambda^{k-1} T^k; \qquad \text{for } \lambda = 1, \quad T^{(1)} x \equiv x^*, \quad \forall x.$$

These mappings correspond to using state-independent coefficients.

This paper will focus mostly on mappings $T^{(\mathbf{c})}$ with state-dependent coefficients $\{c_{ik}\}$. One such example is the λ-weighted Bellman mappings with state-dependent λ values: with $\lambda_i \in [0,1]$,

$$T_i^{(\mathbf{c})} = T_i^{(\lambda_i)}, \qquad i = 1, \ldots, n.$$

In other words, for states i with $\lambda_i < 1$,

$$T_i^{(\mathbf{c})} = (1-\lambda_i)\sum_{k\geq 1} \lambda_i^{k-1} T_i^k, \tag{2.4}$$

and for states i with $\lambda_i = 1$,

$$T_i^{(\mathbf{c})} x \equiv x_i^*, \quad \forall\, x. \tag{2.5}$$

This and other examples will be discussed in later sections, where the coefficients $\mathbf{c}$ can be determined indirectly by the APE algorithms.

Intuitively speaking, our motivation is to use in APE "different kinds" of Bellman mappings for different states. In exact solution methods, as far as the solution is concerned, it does not matter which equation we solve: the equation $x = Tx$ or $x = T^m x$ or $x = T^{(\mathbf{c})} x$. This is different in approximation methods. When we approximate x^* by a vector from some given set, for example, we may get quite different solutions if we solve the problem $x \approx Tx$, i.e.,

$$x_i \approx T_i x, \qquad i = 1, \ldots, n, \tag{2.6}$$

or solve the problem

$$x_i \approx x_i^*, \quad i \in E, \quad \text{and} \quad x_i \approx T_i x, \quad i \notin E, \qquad \text{where } E \subset \{1, \ldots, n\}. \tag{2.7}$$

While this difference is subject to many factors, let us compare the two approximation problems themselves. In the first approximation problem based on (2.6), we use only the relation between states described by the one-step mappings T_i. In the second approximation problem based on (2.7), we combine the information of the true costs x_i^* at some states with the relation conveyed by T_i for some other states. Computationally, it is in general much more expensive to get estimates of x_i^* than estimates of T_i, when the MDP model is unknown and the approximation problems have

to be formed from observations of the Markov chain. However, since we are ultimately interested in finding x with $x_i \approx x_i^*$ for all i, we can argue that we may potentially obtain better results by spending computational efforts unevenly across states, as in problem (2.7). The mapping $T^{(\mathbf{c})}$ used in (2.7) is still extreme: $c_{ik} = 1$ for either $k = \infty$ or $k = 1$. With more general weights c_{ik}, we can form more flexible approximation problems,

$$x_i \approx \textstyle\sum_{k\geq 1} c_{ik} T_i^k x, \qquad i = 1, \ldots, n.$$

They allow the approximate costs to be influenced by different types of information for different states: for instance, for some states, their relation with "nearby" states, and for other states, primarily the estimates of their long-term costs.

Let us introduce now the approximation framework. We consider in this paper APE methods that solve projected versions of weighted Bellman equations (2.3). They are of the form

$$x = \Pi\, T^{(\mathbf{c})} x,$$

where Π is a projection onto a given approximation subspace $S \subset \Re^n$. In the literature the projection Π is commonly defined for a Euclidean norm. We extend the formulation to define Π with respect to a semi-norm. This enhances the range of applications of our model, as we will explain shortly.

For a nonnegative vector $\xi = (\xi_1, \ldots, \xi_n)$, let $\|\cdot\|_\xi$ denote the weighted Euclidean semi-norm on $\Re^n$ given by $\|x\|_\xi^2 = \sum_{i=1}^n \xi_i\, x_i^2$. When ξ is such that for all x, the solution to the problem $\min_{y\in S} \|x - y\|_\xi$ is unique, we can define a (linear) projection mapping Π_ξ onto the subspace S by

$$\Pi_\xi x = \arg\min_{y\in S} \|x - y\|_\xi, \qquad \forall\, x.$$

The preceding condition on ξ is equivalent to[1]

$$\nexists\ y \in S \ \text{ with } y \neq 0,\ \|y\|_\xi = 0, \tag{2.8}$$

or in other words, if $x, y \in S$ and $x_i = y_i$ for all i with $\xi_i \neq 0$, then $x = y$. (Whether this condition holds depends only on which components of ξ are zero.) With Π_ξ thus defined, a projected version of Eq. (2.3) is

$$x = \Pi_\xi\, T^{(\mathbf{c})} x. \tag{2.9}$$

If the projected equation has a unique solution $\bar{x}$, we can use $\bar{x}$ as an approximation of x^*.

We introduce a regularity condition in accordance with the preceding discussion. In this paper, we will focus on projected Bellman equations of the form (2.9) that satisfy this condition.

Definition 2.1 (Regularity Condition for $(\xi, \mathbf{c}, S)$)**.** *A nonnegative vector $\xi \in \Re^n$ with $\sum_{i=1}^n \xi_i = 1$, a collection $\mathbf{c}$ of coefficients of the form (2.1), and a subspace S are said to satisfy the Regularity Condition, if condition (2.8) holds and the projected equation (2.9) has a unique solution.*

One case where the Regularity Condition is always satisfied is approximation by state aggregation. As will be shown in Section 4, a wide class of such APE methods can be viewed as projected equation-based methods with semi-norm projection, and the associated projected equations have a unique solution due to their model approximation nature.

Our main motivation for introducing semi-norm projection in the preceding formulation is as follows. In the context of learning or simulation, the projected equation is formed by the statistics of state transitions and multistage costs from various initial states, and ξ_i is often related to the frequency of including the associated estimates for state i during this procedure. It is natural that algorithms do not gather such information for all states, and consequently, ξ_i is zero for some states. Two common reasons are:

[1] It can be verified that this condition is also equivalent to the following: the matrix $\Phi'\Xi\Phi$ is invertible, where Φ is a (any) $n \times d$ matrix whose columns form a basis of S and $\Xi = diag(\xi)$, the diagonal matrix with ξ_i as the diagonal elements. Furthermore, a matrix representation of Π_ξ is given by $\Phi(\Phi'\Xi\Phi)^{-1}\Phi'\Xi$ for any such Φ.

(i) Some states are never encountered; for example, they may be transient.

(ii) Interpolation/extrapolation schemes are used in approximation (such as in state aggregation). Only the approximate costs at a certain subset of states need to be determined; they in turn determine those at the rest of the states.

By allowing ξ to have zero components, one can deal with the case where in spite of certain states being "neglected" (deliberately or unintentionally), the approximations produced are still meaningful for *all* states. By contrast, if ξ has to be positive, such cases will be excluded from consideration. For example, it would have been impossible to make the connection between aggregation methods and projected equations, without semi-norm projection.

2.2 Approximation Properties

We already know several approximation properties of projected Bellman equations, based on earlier studies on TD methods and generic projected equation methods (see e.g., [TV97, YB10, Sch10]). They provide the theoretical grounds for APE algorithms that solve projected equations (2.9) involving weighted Bellman mappings. Therefore, before proceeding to show that such APE algorithms are both natural and practical (Sections 2.3 and 3), we first summarize the key approximation properties of the projected equations (2.9).

First, we note that since the mapping $T^{(\mathbf{c})}$ is affine, it can be expressed as

$$T^{(\mathbf{c})}x = Ax + b$$

for some (substochastic) matrix A and vector b, and the Regularity Condition implies that the matrix $(I - \Pi_\xi A)$ is invertible. This representation will appear frequently in the subsequent analysis.

Under the Regularity Condition, the equation $x = \Pi_\xi T^{(\mathbf{c})}x$ is well-defined and has a unique solution $\bar{x} \in S$. If $x^* \in S$, then $x^* = \Pi_\xi x^* = \bar{x}$. In general, $\bar{x}$ is a biased approximation of x^* in the sense that $\bar{x}$ may differ from Πx^* (the Euclidean projection of x^* onto S) and may also differ from $\Pi_\xi x^*$ (the weighted Euclidean projection of x^* on S).

Geometric Interpretation

Scherrer [Sch10] showed a geometric interpretation of $\bar{x}$ as an *oblique projection* of x^*, which holds also when $\|\cdot\|_\xi$ is a semi-norm:[2]

[2]We describe briefly the derivation. First, for any two d-dimensional subspaces S, M of $\Re^n$ such that no vector in M is orthogonal to S, there is an associate oblique projection operator $\tilde{\Pi}$ defined by

$$\tilde{\Pi}x \in S, \qquad x - \tilde{\Pi}x \perp M, \qquad \forall\, x.$$

A matrix representation of $\tilde{\Pi}$ is given by

$$\tilde{\Pi} = \Phi(W'\Phi)^{-1}W',$$

where Φ and W are matrices whose columns form a basis of S and M, respectively (see Saad [Saa03, Chap. 1.12]). The oblique projection interpretation of $\bar{x}$ under the Regularity Condition can then be seen as follows. Since $\bar{x} = \Pi_\xi(A\bar{x}+b)$ and $b = x^* - Ax^*$,

$$\bar{x} = (I - \Pi_\xi A)^{-1}\Pi_\xi(I - A)x^*.$$

Expressed in terms of Φ, $(I - \Pi_\xi A)^{-1}\Pi_\xi = \Phi(\Phi'\Xi(I - A)\Phi)^{-1}\Phi'\Xi$, where $\Xi = diag(\xi)$, so

$$\bar{x} = \Phi(\Phi'\Xi(I - A)\Phi)^{-1}\Phi'\Xi(I - A)x^*.$$

For a derivation of the preceding statement, see e.g., [YB10, Lemma 2.1]. One can also verify the statement by working with the projected Bellman equation with the change of variable $x = \Phi r$: $\Phi r = \Pi_\xi(A\Phi r + b)$, and using the matrix representation of Π_ξ as discussed in Footnote 1. In the above, the matrix $\Phi'\Xi(I - A)\Phi$ is invertible since the matrix $I - \Pi_\xi A$ is invertible. (In fact, the two are equivalent: one matrix is invertible if and only if the other is.) Then, from the matrix representation of an oblique projection operator $\tilde{\Pi}$, we recognize that in the above expression of $\bar{x}$, the linear mapping appearing before x^* is the oblique projection operator with respect to S and M, with M being the column space of $(I - A)'\Xi\Phi$, equivalently, the image of S under the linear transformation $(I - A)'\Xi$.

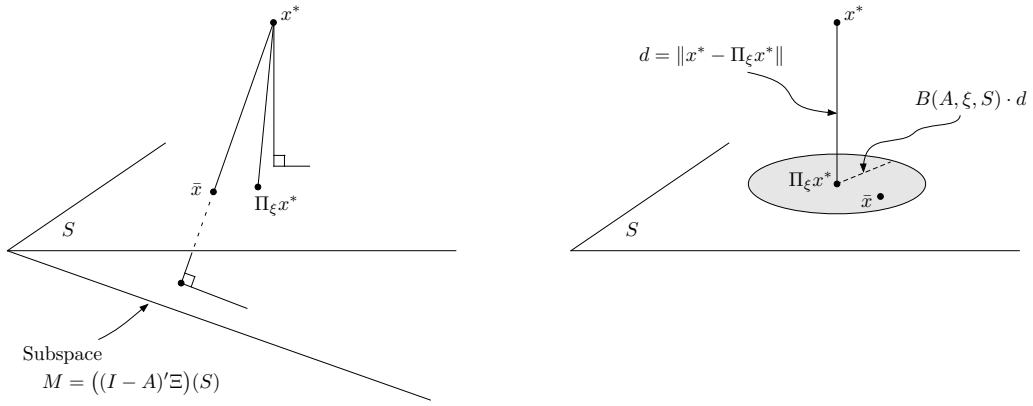

Figure 1: Left: A geometric relation between $\bar{x}$ and x^*. The point $\bar{x}$ is an oblique projection of x^* onto S with respect to a subspace M that is the image of a linear transformation of S. Right: An illustration of the bound (2.11) on approximation bias. The distance $\|\bar{x} - \Pi_\xi x^*\|$ is no greater than $B(A, \xi, S) \cdot \|x^* - \Pi_\xi x^*\|$, where like the subspace M, the factor $B(A, \xi, S)$ depends on S, ξ and the matrix A in $T^{(\mathbf{c})}$.

Proposition 2.1 (An oblique projection interpretation of $\bar{x}$ [Sch10])**.** *Under the Regularity Condition, $\bar{x}$ is the projection of x^* onto S and orthogonal to a subspace M of the same dimensionality as S:*

$$\bar{x} \in S, \qquad x^* - \bar{x} \perp M.$$

This subspace M is the image of S under the linear transformation $(I-A)'\Xi$, where Ξ is the diagonal matrix with ξ on its main diagonal.

Note that the subspace M depends critically on A, the matrix in the equation $x = T^{(\mathbf{c})}x$ we want to solve. Conceptually, the oblique projection view provides us with the understanding of the source of the bias in $\bar{x}$; see Fig. 1 (left).

Approximation Error Bounds

When $\|\cdot\|_\xi$ is a norm, several approximation error bounds are available, which relate the bias $\bar{x} - \Pi_\xi x^*$ to the distance from x^* to the subspace S. These bounds extend to the case where $\|\cdot\|_\xi$ is a semi-norm, as described below. Some derivation details are given in Appendix A.

Under the Regularity Condition, as pointed out in [YB10], we have[3]

$$\bar{x} - \Pi_\xi x^* = -(I - \Pi_\xi A)^{-1}\Pi_\xi A(I - \Pi_\xi)(x^* - \Pi_\xi x^*) \tag{2.10}$$

and hence the following bound.

Proposition 2.2 ([YB10])**.** *Under the Regularity Condition, for any norm $\|\cdot\|$ on $\Re^n$ and its associated matrix norm, we have*

$$\|\bar{x} - \Pi_\xi x^*\| \le B(A, \xi, S) \cdot \|x^* - \Pi_\xi x^*\|, \tag{2.11}$$

[3]Equation (2.10) can be derived as follows. Since $\bar{x} = \Pi_\xi(A\bar{x} + b)$ and $\Pi_\xi x^* = \Pi_\xi(Ax^* + b)$, we have

$$\bar{x} - \Pi_\xi x^* = \Pi_\xi A(\bar{x} - x^*) = \Pi_\xi A(\bar{x} - \Pi_\xi x^*) + \Pi_\xi A(\Pi_\xi x^* - x^*),$$

and hence

$$\bar{x} - \Pi_\xi x^* = (I - \Pi_\xi A)^{-1}\Pi_\xi A(\Pi_\xi x^* - x^*).$$

Since $\Pi_\xi(\Pi_\xi x^* - x^*) = 0$, Eq. (2.10) follows.

where

$$B(A,\xi,S) = \|(I-\Pi_\xi A)^{-1}\Pi_\xi A(I-\Pi_\xi)\|.$$

Consistently with the oblique projection view, the bound shows that the approximation bias depends on the relation between S, ξ and the matrix A in $T^{(\mathbf{c})}$.[4] See Fig. 1 (right) for an illustration of the bound.

The next few bounds all measure the approximation bias using the semi-norm $\|\cdot\|_\xi$. They are useful when we care only about the approximation at those states i with $\xi_i > 0$. For an $n \times n$ matrix L, let us define

$$\|L\|_\xi = \sup_{y:\|y\|_\xi \le 1} \|Ly\|_\xi.$$

It is possible that $\|L\|_\xi = +\infty$ since $\|\cdot\|_\xi$ is a semi-norm. The first bound is analogous to (2.11) and follows from Eq. (2.10). The proof can be found in Appendix A.

Proposition 2.3. *Under the Regularity Condition,*

$$\|\bar{x} - \Pi_\xi x^*\|_\xi \le \tilde{B}(A,\xi,S) \cdot \|x^* - \Pi_\xi x^*\|_\xi, \tag{2.12}$$

where

$$\tilde{B}(A,\xi,S) = \|(I-\Pi_\xi A)^{-1}\Pi_\xi A(I-\Pi_\xi)\|_\xi.$$

The bound (2.12) is non-vacuous when $\tilde{B}(A,\xi,S) < \infty$, and this is true if $\|\Pi_\xi A\|_\xi < \infty$, and in particular, if ξ is an invariant distribution of P.

The bound (2.12) implies a case where x^* is partially obtained by solving the projected equation:

Corollary 2.1. *Let the Regularity Condition hold. If $\|x^* - \Pi_\xi x^*\|_\xi = 0$ and $\|\Pi_\xi A\|_\xi < \infty$, then $\bar{x} = \Pi_\xi x^*$, so*

$$\|\bar{x} - x^*\|_\xi = 0.$$

In other words, $\bar{x}$ coincides with x^* at states i with $\xi_i > 0$, if S contains a vector which coincides with x^* at these states and $\|\Pi_\xi A\|_\xi < \infty$. This can also be shown directly; see Appendix A.

The conclusions of Prop. 2.3 and Cor. 2.1 can, in fact, be made more general. The semi-norm $\|\cdot\|_\xi$ in their statements can be replaced by any weighted Euclidean semi-norm $\|\cdot\|_{\hat{\xi}}$ such that $\hat{\xi}$ has the same zero components as ξ. (These semi-norms are equivalent in some sense.)

We mention three more bounds, which rely on a contraction-type argument of Tsitsiklis and Van Roy [TV97], and apply under certain conditions.

Proposition 2.4 ([TV97])**.** *Let the Regularity Condition hold. Suppose that $\|\Pi_\xi A\|_\xi < 1$. Then*

$$\|\bar{x} - \Pi_\xi x^*\|_\xi \le \frac{\alpha}{\sqrt{1-\alpha^2}} \|x^* - \Pi_\xi x^*\|_\xi, \qquad \textit{where} \quad \alpha = \|\Pi_\xi A\|_\xi. \tag{2.13}$$

A main application of the bound (2.13) is in the following two cases. There the condition $\|\Pi_\xi A\|_\xi < 1$ holds and furthermore, $\alpha = \|\Pi_\xi A\|_\xi$ in the bound can be replaced by a simpler expression.

[4] For large-scale problems with unknown model parameters, it is not easy to compute the factor $B(A,\xi,S)$ in (2.11), however.

Corollary 2.2. *Let the Regularity Condition hold. Suppose that ξ is an invariant distribution of P and the coefficients $\mathbf{c}$ in $T^{(\mathbf{c})}$ satisfy*

$$\sum_{k\geq 1} \bar{c}_k \beta^{2k} < 1, \qquad \text{where } \bar{c}_k = \max_{1\leq i\leq n} c_{ik}.$$

Then

$$\|\Pi_\xi A\|_\xi \leq \bar{\alpha} = \sqrt{\textstyle\sum_{k\geq 1} \bar{c}_k \beta^{2k}}, \tag{2.14}$$

so the bound (2.13) holds with $\bar{\alpha}$ in place of α.

Without conditions on c_{ik}, it is not always true that $\|\Pi_\xi A\|_\xi < 1$ when ξ is an invariant distribution of P. But for weighted Bellman mappings with state-independent coefficients, this is always the case and the bound (2.14) can be strengthened as follows.

Corollary 2.3. *Let the Regularity Condition hold. Suppose that ξ is an invariant distribution of P and the coefficients $\mathbf{c}$ in $T^{(\mathbf{c})}$ are state-independent: $c_{ik} = c_k$ for all i. Then*

$$\|\Pi_\xi A\|_\xi \leq \hat{\alpha} = \textstyle\sum_{k\geq 1} c_k \, \beta^k \leq \beta, \tag{2.15}$$

so the bound (2.13) holds with $\hat{\alpha}$ in place of α.

Some familiar examples of this bound are: for $T^{(\mathbf{c})} = T^m$, $\hat{\alpha} = \beta^m$, and for $T^{(\mathbf{c})} = T^{(\lambda)}$, $\hat{\alpha} = (1-\lambda)\beta/(1-\lambda\beta)$.

We make a few remarks. In general, the contraction-based bounds are less sharp than the bound (2.11).[5] It is also true that all the bounds involve relaxations and hence do not necessarily characterize well the approximation error (see Example 2.1 for a comparison of these bounds with the bias-to-distance ratios in a given problem). Despite its being easy to compute, the bound (2.14) is unsatisfactory because it neglects the dependency of the coefficients c_{ik} on the states. Example 2.1 below illustrates that using $T^{(\mathbf{c})}$ with state-dependent weights can be beneficial. The example also shows that the case $\|\Pi_\xi A\|_\xi > 1$ need not result in an inferior cost approximation.

Example 2.1. The MDP problem is from [Ber95] (see also [BT96, Example 6.5]), for which the TD(0) solution is known to have a large approximation bias. The Markov chain has 51 states $\{0, 1, \ldots 50\}$ and deterministic transitions from state i to state $i-1$, with state 0 being absorbing. The one stage costs are 1 except at states 0 and 50, which are 0 and -49 respectively. The discount factor is 0.999, the subspace S is one-dimensional: $S = \{x \mid x_i = c\, i, i = 0, \ldots, 50, \text{ for some } c \in \Re\}$, and the projection is with respect to the unweighted Euclidean norm. We use the Bellman mappings given by Eq. (2.4) with state-dependent λ_i parameters (their associated projected equations can be efficiently computed; see Section 3.4.1). We let λ_i be nonzero for only a few selected states i: in one case (Approximation I), $\lambda_i = 0.9$ for only states 40-50, and in another case (Approximation II), $\lambda_i = 0.9$ for only states 45-50. As shown in Fig. 2, despite having $\lambda_i = 0$ for most of the states, compared to the TD(0) solution, the approximation bias is significantly reduced.

We calculate the approximation error bounds for TD(0), TD(0.9) and the two approximations mentioned above. In this case, $\xi = (1, \ldots, 1)/51$ is not an invariant distribution of P. For both TD(0) and TD(0.9), $\|\Pi_\xi A\|_\xi < 1$. For both approximations with state-dependent λ parameters, $\|\Pi_\xi A\|_\xi > 1$, so the contraction-based bound (2.13) is inapplicable. We measure the bias with respect to $\|\cdot\|_\xi$, and we list below the true relative bias and the factors in the error bounds (2.13), (2.11):

[5] In the case where $\xi > 0$ and $\|\cdot\|_\xi$ is the norm for measuring approximation error, it is shown in [YB10] that the bound (2.11) is attained by a worst-case choice of the vector b in the Bellman equation $x = T^{(\mathbf{c})}x = Ax + b$, and when $\|\Pi_\xi A\|_\xi < 1$, (2.11) is tighter than the contraction-based bound (2.13).

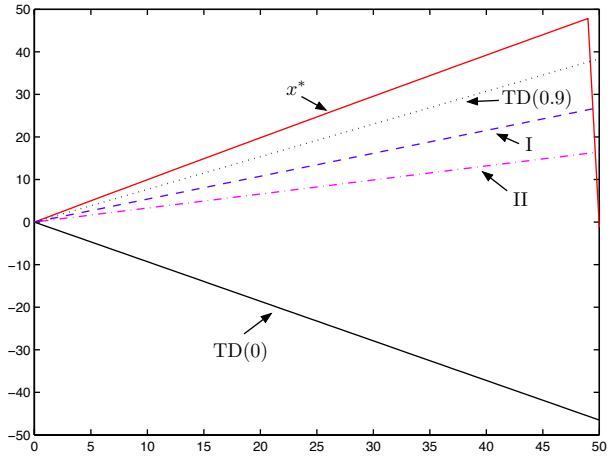

Figure 2: Example of approximation bias reduction using weighted Bellman mappings with state-dependent weights. The X-axis represents the states 0-50 in the problem. Plotted for comparison are: the cost function x^*, the TD(0) and TD(0.9)-approximation, and approximations I and II, which are obtained by using the Bellman mapping (2.4) with state-dependent λ_i parameters, where $\lambda_i = 0$ at all but a few selected states. In I, $\lambda_i = 0.9$ for states 40-50; in II, $\lambda_i = 0.9$ for states 45-50.

	TD(0)	TD(0.9)	Approximation I	Approximation II
$\|\bar{x} - \Pi_\xi x^*\|_\xi / \|x^* - \Pi_\xi x^*\|_\xi$	7.865	0.6608	1.633	2.519
$\alpha/\sqrt{1-\alpha^2}$	22.34	3.434	n/a	n/a
$B(A, \xi, S)$	7.879	2.261	4.145	5.572

In the above the bounds are calculated using the knowledge of A and ξ. We note that in practice, computing $\alpha = \|\Pi_\xi A\|_\xi$ and $B(A, \xi, S)$ is equally hard when the model of the problem is not available. □

Discussion on Choices of Weights

To apply weighted Bellman mappings in APE, the central issues besides computation are how to choose S and ξ, and which mapping $T^{(\mathbf{c})}$ to use so that we can balance the approximation bias against the estimation variance, making algorithms overall efficient. These issues also arise in TD methods, and they are difficult. At present, we lack reliable criteria and tractable ways to optimize the choice of S, ξ and $\mathbf{c}$.

Some intuition may serve as our guide, however, regarding the choice of $T^{(\mathbf{c})}$. Let us view the coefficients $\{c_{ik} \mid k = 1, 2, \ldots\}$ as a probability distribution on the numbers of stages starting from state i. From the system of equations $x = \Pi_\xi T^{(\mathbf{c})} x$, we see intuitively that by putting a larger probability on later stages for a state j and states near it (in the transition graph), we rely more on direct estimates of long-term costs from j and less on the relation between j and other states, for cost approximation at state j. Suppose there are a few "sensitive" states j where the cost approximation accuracy influences the overall approximation quality greatly. Then we can choose $\{c_{jk}\}$ in the aforementioned manner, assigning a large probability on later stages for these and their nearby states. For the rest of the states i, their policy costs may now be more reliably inferred from the relation between i and the nearby states, so we can choose $\{c_{ik}\}$ to put the majority of probability mass on earlier stages. (See Example 2.1 for a demonstration of this point.) This also means that by choosing the state-dependent weights $\{c_{ik}\}$ properly, approximation bias can be potentially reduced with limited increase in estimation variance.

2.3 Computational Aspects: A Generic APE Algorithm

We now consider computational aspects of using weighted Bellman equations in learning and simulation based APE. We show that a natural, computationally straightforward way of doing APE leads to such equations. This APE algorithm will be described in a generic form, to highlight the connection with weighted Bellman mappings and projection, and to leave out details, such as data collection mechanisms and probabilistic assumptions, which are not of interest at this point. This generic APE algorithm will serve as a "template" for the example algorithms in the next section.

We begin with several equivalent forms of the projected Bellman equation $x = \Pi_\xi T^{(\mathbf{c})} x$. They are more convenient from the computational perspective, especially for learning and simulation-based computation where ξ and $\mathbf{c}$ can be implicitly determined by the APE algorithms. For $x, y \in \Re^n$, we denote by $\langle x, y \rangle$ the usual inner product of x and y, and for a nonnegative vector $\xi \in \Re^n$, we denote $\langle x, y \rangle_\xi = \sum_{i=1}^n \xi_i x_i y_i$. We can write the projected Bellman equation $x = \Pi_\xi T^{(\mathbf{c})} x$ equivalently as

$$x \in S, \qquad \Pi_\xi (x - T^{(\mathbf{c})} x) = 0,$$

or

$$x \in S, \qquad \big\langle v \, , \, x - T^{(\mathbf{c})} x \big\rangle_\xi = 0, \qquad \forall \, v \in V_S, \tag{2.16}$$

where V_S is a finite set of vectors that span the subspace S (see property (b) in Appendix A). Equivalently, with $\Xi = diag(\xi)$,

$$x \in S, \qquad \big\langle v \, , \, \Xi (x - T^{(\mathbf{c})} x) \big\rangle = 0, \qquad \forall \, v \in V_S, \tag{2.17}$$

i.e., $x \in S$ and $\Xi(x - T^{(\mathbf{c})} x) \perp S$.

Consider now a generic APE algorithm, which constructs a sequence of equations analogous to Eq. (2.17). At each iteration t, it

- collects sequences of state transitions and transition costs,
- defines an $\Re^n$-valued affine function $G_t(x)$ based on the data, and
- builds a system of linear equations[6] analogous to Eq. (2.17):

$$x \in S, \qquad \big\langle v \, , \, G_t(x) \big\rangle = 0, \qquad \forall \, v \in V_S. \tag{2.18}$$

Here the function $G_t = (G_{t,1}, \ldots, G_{t,n})$ takes a particular form: for each i, $G_{t,i}$ is a weighted sum of $N_i(t)$ functions (none if $N_i(t) = 0$):

$$G_{t,i}(x) = \textstyle\sum_{m=1}^{N_i(t)} \, w_{im}(t) \big(x_i - \tilde{T}_{im,t} \, x \big) \tag{2.19}$$

with $N_i(t) \geq 0$ and with $w_{im}(t) \geq 0$ being the weights. For each (i, m), $\tilde{T}_{im,t} : \Re^n \to \Re$ is an affine function specified by the data. It will be defined to resemble a multistep Bellman mapping in the example algorithms to follow; the precise definition is not needed for now. The data-determined function terms $x_i - \tilde{T}_{im,t} \, x$ may be viewed as the "building-blocks" of the algorithm.

We relate the generic APE algorithm to projected equations with weighted Bellman mappings by relating the functions $G_{t,i}$ to scaled, approximate versions of some affine functions

$$x_i - T_i^{(\mathbf{c})} x, \qquad i = 1, \ldots, n,$$

[6] Storing Eq. (2.18) is memory-efficient. By making the change of variable $x = \Phi r$, where Φ is a matrix whose columns span S and r is a column vector, Eq. (2.17) is equivalently $\langle v \, , \, G_t(\Phi r) \rangle = 0, v \in V_S$. So we only need to store $|V_S|$ affine functions on the low-dimensional space of r, $g_{v,t}(r) = \langle v \, , \, G_t(\Phi r) \rangle$.

roughly speaking. More precisely, this is done by averaging and normalization. Let $H_t : \Re^n \to \Re^n$ be an affine mapping with its ith component given by

$$H_{t,i}\, x = \textstyle\sum_{m=1}^{N_i(t)} \bar{w}_{im}(t) \cdot \tilde{T}_{im,t}\, x, \quad \forall\, x,$$

where $\bar{w}_{im}(t) = \frac{w_{im}(t)}{w_i(t)}$ and $w_i(t) = \sum_{m=1}^{N_i(t)} w_{im}(t)$. Also let $w(t) = \sum_{i=1}^{n} w_i(t)$ and

$$\xi_i(t) = \tfrac{w_i(t)}{w(t)}, \qquad\qquad \xi(t) = \big(\xi_1(t),\, \ldots\, ,\, \xi_n(t)\big).$$

In the above, we treat $0/0$ as 0. Then we have

$$\begin{aligned} G_{t,i}(x) &= w(t) \cdot \xi_i(t) \cdot (x_i - H_{t,i}x), \qquad i = 1, \ldots, n, \\ \langle v\, ,\, G_t(x)\rangle &= w(t) \cdot \langle v\, ,\, x - H_t x\rangle_{\xi(t)}, \qquad \forall\, v \in V_S, \end{aligned}$$

so Eq. (2.18) is equivalent to

$$x \in S, \qquad \big\langle v\, ,\, x - H_t x\big\rangle_{\xi(t)} = 0, \quad \forall\ v \in V_S. \tag{2.20}$$

Comparing Eq. (2.20) with (2.16), the following conclusion is then immediate.

Suppose there exist some vector $\xi = (\xi_1, \ldots, \xi_n) \geq 0$ and some set $\mathbf{c}$ of sequences satisfying condition (2.1), such that for each $i = 1, \ldots, n$,

$$\xi_i(t) \to \xi_i, \qquad \|\xi_i(t) H_{t,i} - \xi_i T_i^{(\mathbf{c})}\| \to 0, \qquad \text{as} \ \ t \to \infty. \tag{2.21}$$

Then, asymptotically, Eq. (2.20) tends to the equation

$$x \in S, \qquad \big\langle v\, ,\, x - T^{(\mathbf{c})} x\big\rangle_{\xi} = 0, \quad \forall\ v \in V_S.$$

When ξ satisfies condition (2.8), this equation is the same as $x = \Pi_\xi\, T^{(\mathbf{c})} x$ [cf. Eq. (2.16)]. Suppose in addition that it has a unique solution $\bar{x}$; in other words, suppose $(\xi, \mathbf{c}, S)$ satisfies the Regularity Condition. Then the solutions of the equations (2.18) from the generic APE algorithm approach $\bar{x}$ asymptotically.

The above discussion links the generic APE algorithm to the projected equation method. There are other reasons to describe a projected equation-based algorithm in the form of (2.18)-(2.19) (instead of least-squares optimization problems, for instance). Let us discuss them in preparation for the example algorithms to be introduced shortly. In this formulation:

(i) We have all data-related terms appearing as the arguments of the usual inner products $\langle v, \cdot\rangle, v \in V_S$, which are data-independent [cf. Eq. (2.18)]. This emphasizes the simple relation between the data, the subspace S, and the entire system of equations in the approximation scheme. It is also computationally expedient for algorithms to work with G_t, because there is no need to actually normalize G_t, and also because ξ and $\mathbf{c}$ are often jointly determined in the algorithms.

(ii) The focus is on the components of Bellman mappings for each individual state. This helps design and compare algorithms at a high level. In most cases, efficient implementation of the algorithms can then be obtained via certain well-known technique or routine procedure (such as the one given in Footnote 6). By comparison, if one were to work directly with sophisticated computation formulas (such as those for TD(λ)-type algorithms), the relation of the algorithm with the Bellman mapping and projection could be much less obvious, which could make it difficult to see how to modify the formulas in order to modify the associated Bellman mapping and projection in a desired way. (As an example, from Eqs. (2.18) and (2.19) it is clear that we can set ξ_i to zero by excluding all the terms $(x_i - \tilde{T}_{im,t})$ associated with state i. By contrast, from the recursive formulas of a TD(λ) algorithm, one cannot tell immediately how to achieve the same effect if one is unfamiliar with the "high-level equation" that gives rise to the formulas in the first place.)

3 Some Example Algorithms with Learning and Simulation

In this section we give several algorithms as examples of the generic APE algorithm outlined in Section 2.3. The convergence condition (2.21) holds for them due to the way data – trajectories of states – are generated and used. In all the examples, we accumulate data over time and based on the most recent information, we add terms to $G_{t-1}(x)$ of the previous equation to form $G_t(x)$ of the current equation [cf. Eq. (2.18)]. We discuss the simplest example first and then its more general variants in Section 3.2. In Sections 3.3 and 3.4, we introduce algorithms of TD(λ)-type.

As before, our focus will be on the equations constructed by the algorithms and their connection with projected weighted Bellman equations. We will not discuss how the equations are to be solved (they can be solved exactly or inexactly, and a variety of solvers may be used). We start by specifying the simulation model and some definitions.

3.1 Notation and Specifications

State Trajectories and Associated Mappings

We use a simple data generation scheme in all the examples. At iteration t, a state trajectory $\mathcal{I}_t = (i_0, i_1, \ldots, i_\ell)$ is generated, and its length is random but bounded: $\ell \leq N$ for some given positive integer N. This trajectory is generated in the following manner:

- First, the starting state i_0 and the number of transitions, $\ell \geq 1$, are chosen according to some given conditional probability distribution $\Gamma\big((i_0, \ell) \mid \mathcal{I}_{t-1}\big)$, conditional on the trajectory $\mathcal{I}_{t-1}$ of the previous iteration.[7]
- Next, the state transitions $(i_k, i_{k+1}), k < \ell$, are generated, by a simulator or the learning environment, according to some transition probabilities $\{\bar{p}_{ij}\}$.

The (online) learning context differs from the simulation context in that it is not as easy to obtain trajectories from desirable initial states as when a simulator is available, and instead, i_0 has to be a state of the environment that one encounters. Nevertheless, the situation can be described by a suitably chosen Γ.

For the generation of state transitions, we consider two cases:

- Ordinary PE, where transitions (i_k, i_{k+1}) are generated according to the transition probabilities $\{p_{ij}\}$ under the policy μ that we aim to evaluate (i.e., $\bar{p}_{ij} = p_{ij}$ for all states i, j).
- Exploration-enhanced PE, where the transition probabilities $\bar{p}_{ij}$ differ from those under μ.

The state trajectories, $\mathcal{I}_t, t \geq 1$, thus generated form a finite-state Markov chain. See Fig. 3 for an illustration.

For a state sequence $\mathcal{I} = (i_0, \ldots, i_\ell)$, we define a sampled version of a multistep Bellman mapping, $\tilde{T}_{\mathcal{I}} : \Re^n \to \Re$, depending on how $\mathcal{I}$ is generated. In the case of ordinary PE, we define

$$\tilde{T}_{\mathcal{I}}\, x = \textstyle\sum_{k=0}^{\ell-1} \beta^k g_\mu(i_k, i_{k+1}) + \beta^\ell\, x_{i_\ell}, \qquad \forall\, x, \tag{3.1}$$

where $g_\mu(i, j)$ denote the cost of transition from state i to j under the policy μ. In the case of exploration-enhanced PE,[8] $\tilde{T}_{\mathcal{I}}$ is defined to compensate for the difference between $\bar{p}_{ij}$ and the transition probabilities under μ:

$$\tilde{T}_{\mathcal{I}}\, x = \textstyle\sum_{k=0}^{\ell-1} \beta^k\, \rho_{k+1} \cdot g_\mu(i_k, i_{k+1}) + \beta^\ell\, \rho_\ell \cdot x_{i_\ell}, \qquad \forall\, x, \tag{3.2}$$

[7] The conditional distribution Γ can be specified indirectly and here is an example. We first choose the initial state for $\mathcal{I}_t$: with a certain probability that depends on $\mathcal{I}_{t-1}$, we let i_0 be the last state of the trajectory $\mathcal{I}_{t-1}$, thereby continuing that trajectory, whereas with a certain probability, we generate i_0 according to a given distribution, thereby starting a new trajectory. We then choose the number ℓ according to some given distribution that depends on i_0.

[8] More precisely, in exploration-enhanced PE, states i here need to be associated with state-action pairs in the MDP. For notational simplicity, we do not introduce this association in the paper.

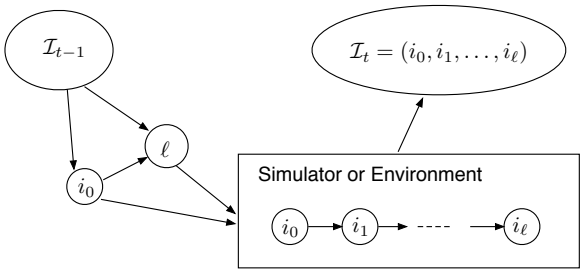

Figure 3: The procedure of generating the state trajectory $\mathcal{I}_t$ for iteration t in the APE Algorithms. Direct edges indicate some of the dependence relations between the variables.

where

$$\rho_{k+1} = \prod_{j=0}^{k} \frac{p_{i_j i_{j+1}}}{\bar{p}_{i_j i_{j+1}}}, \qquad k \geq 0,$$

and it is assumed that $\{\bar{p}_{ij}\}$ satisfy that for all states i, j, $\bar{p}_{ij} > 0$ if $p_{ij} > 0$.

The examples will be meant for either ordinary PE or exploration-enhanced PE. To avoid treating the two cases separately, we adopt the convention that the mapping $\tilde{T}_{\mathcal{I}}$ appearing in the examples is defined according to the corresponding Eq. (3.1) or (3.2).

Notation

We introduce some notation to be used throughout the paper. For a state sequence $\mathcal{I} = (i_0, \ldots, i_\ell)$, we denote by $|\mathcal{I}|$ the number of state transitions and by $(\mathcal{I})_j$ the jth state: $|\mathcal{I}| = \ell$ and $(\mathcal{I})_j = i_j$. We also denote $(\mathcal{I})_j^m = (i_j, i_{j+1}, \ldots, i_m)$, the segment of $\mathcal{I}$ between i_j and i_m. With this notation, for example, $\tilde{T}_{(\mathcal{I})_j^m}$ is the mapping given by Eq. (3.1) or (3.2) with the segment $(\mathcal{I})_j^m$ in place of $\mathcal{I}$.

We will be dealing with random affine functions defined by data. It is convenient to view these functions as random variables. More precisely, for a positive integer m, let $\mathcal{A}_m[x]$ denote the space of $\Re^m$-valued affine functions of x:

$$\mathcal{A}_m[x] = \big\{Cx + b \mid C \text{ is an } m \times n \text{ matrix}, b \in \Re^m\big\}.$$

We define $\mathcal{A}_m[x]$-valued random variables, their expectation and sample mean in an obvious way. An $\mathcal{A}_m[x]$-valued random variable ψ is a random function $Cx + b$, where the matrix C and vector b are random. The expectation of ψ, which will be denoted $\mathbb{E}[\psi]$ or $\mathbb{E}[\psi(x)]$, is the function

$$\mathbb{E}[\psi](x) = \mathbb{E}[C]\, x + \mathbb{E}[b].$$

If either C or b does not have a finite expectation, then $\mathbb{E}[\psi]$ is undefined, a case that we will not encounter. For a sequence of $\mathcal{A}_m[x]$-valued random variables, $\psi_t(x) = C_t x + b_t$, $t = 1, 2, \ldots$, the sample average at time t is the function $\bar{C}x + \bar{b} \in \mathcal{A}_m[x]$ with $\bar{C} = \frac{1}{t}\sum_{\tau=1}^{t} C_\tau$ and $\bar{b} = \frac{1}{t}\sum_{\tau=1}^{t} b_\tau$, where C_τ, b_τ, $\tau \leq t$, take their realized values. We say that $\{\psi_t\}$ converges to $\psi(x) = Cx + b$ with probability one, if $\{C_t\}$ and $\{b_t\}$ converge to C, b, respectively, with probability one. We will henceforth omit the term "with probability one," for simplicity.

We use δ for the indicator function. To simplify notation, we write $a_k \propto a'_k$ for two nonnegative scalar sequences, $\{a_k\}$, $\{a'_k\}$, such that $\sum_k a'_k > 0$ and $a_k = a'_k / \sum_k a'_k$. We use this notation also when $a'_k = 0$ for all k; then, "$a_k \propto a'_k$" is regarded to be true for any $\{a_k\}$ with $\sum_k a_k = 1$.

3.2 A Simple Algorithm and its Variants

We start with a simple algorithm, Algorithm I. It adds one suitably weighted term to $G_{t-1}(x)$ to form $G_t(x)$ as per Eq. (2.19). Also it uses certain nonnegative scalars α_{ik}, $k = 1, \ldots, N$, for all states i.

These scalars, together with the trajectory generation scheme, determine how terms associated with sequences of different lengths are weighted relative to each other.

Algorithm I: (iteration t)

(1) Choose a starting state i_0 and a number $\ell \geq 1$ according to probability $\Gamma\big((i_0,\ell) \mid \mathcal{I}_{t-1}\big)$. Generate a state trajectory $\mathcal{I}_t = (i_0,\ldots,i_\ell)$.

(2) Define the components of $G_t(x)$ by

$$G_{t,i}(x) = G_{t-1,i}(x) + \delta\{i = i_0\} \cdot \alpha_{i\ell}\left(x_i - \tilde{T}_{\mathcal{I}_t} x\right), \qquad i = 1,\ldots,n.$$

(3) Form a system of equations

$$x \in S, \quad f_{v,t}(x) = 0, \quad \forall\, v \in V_S, \qquad \text{(Equation}(t)\text{)}$$

where $f_{v,t}(x) = \big\langle v\,,\, \frac{G_t(x)}{t}\big\rangle$.

More complex variants can be created by including more function terms in G_t at each iteration and thus extracting more information from the trajectories. For example, at step 2, we may add to $G_{t-1}(x)$ the function terms associated with every segment of the trajectory $\mathcal{I}_t$.

Variant A of Algorithm I: We replace step 2 with

(2′) Define $G_t(x)$ by

$$G_{t,i}(x) = G_{t-1,i}(x) + \sum_{j=0}^{\ell-1} \delta\{i = i_j\} \cdot \sum_{k=1}^{\ell-j} \alpha_{ik}\left(x_i - \tilde{T}_{(\mathcal{I}_t)_j^{j+k}} x\right), \qquad i = 1,\ldots,n.$$

(Recall that $(\mathcal{I}_t)_j^{j+k}$ denotes the segment between i_j and i_{j+k}.) Alternatively, we can be more selective and add terms for only some segments. We can also vary the weights α_{ik} with the trajectory. The TD-type algorithms given in Sections 3.3 and 3.4 have some of the features just mentioned. Here let us discuss some less complicated examples.

If we are given in advance a Bellman mapping $T^{(\mathbf{c})}$ whose coefficients c_{ik} are zero for $k > N$, we can implement an APE that builds a corresponding projected equation $x = \Pi_\xi T^{(\mathbf{c})} x$, in two simple ways. One way is to generate N transitions at every iteration and add at step 2 the function

$$\delta\{i = i_0\} \cdot \textstyle\sum_{k=1}^{N} c_{ik}\left(x_i - \tilde{T}_{(\mathcal{I}_t)_0^k} x\right)$$

to $G_{t-1,i}(x)$ when forming Equation(t) at iteration t. Another way is to view $\{c_{ik}\}$ as probabilities and use Algorithm I with ℓ chosen with probability $c_{i_0\ell}$ and with $\alpha_{ik} = 1$ for all i,k. Both implementations can be carried out in either the simulation or the learning context.

Consider another example. Suppose that at step 2, we include only the function terms associated with those trajectory segments that are sufficiently long and start from certain states. In particular, let $N_0 < N$ be a positive integer, and let

$$\text{List}_i \subset \{1,\ldots,n\}, \qquad i = 1,\ldots,n,$$

be subsets of states associated with each state. (Excluding a state j from all these subsets has the effect of setting $\xi_j = 0$ in the semi-norm $\|\cdot\|_\xi$ for the projection operation.)

Variant B of Algorithm I: We replace step 2 with

(2″) Define $G_t(x)$ by

$$G_{t,i}(x) = G_{t-1,i}(x) + \sum_{j=0}^{\ell-N_0} \delta\{i = i_j,\, i \in \mathrm{List}_{i_0}\} \cdot \alpha_{i(\ell-j)} \left(x_i - \tilde{T}_{(\mathcal{I}_t)_j^\ell}\, x\right), \qquad i = 1, \ldots, n.$$

For actual computation, the equations in the above algorithms are stored in terms of low-dimensional matrices and vectors, which are updated from one iteration to another. For specific formulas, see the discussions in Footnote 6, Section 3.3.1 and Appendix B.

Projected Weighted Bellman Equations in the Limit

Like the generic APE algorithm, in the limit as $t \to \infty$, the equations from the above algorithms can be associated with projected weighted Bellman equations. Moreover, these limiting equations can be expressed in terms of the probability distribution of the trajectories and other algorithmic parameters. The derivation is similar for all the algorithms. Below let us derive the associated projected equations for Algorithm I and its variant B.

Consider first Algorithm I. The derivation is based on the following fact. The way the state sequence $\mathcal{I}_t$ is generated and used ensures that when we average those random mappings $\tilde{T}_{\mathcal{I}_\tau}$, $\tau \le t$, whose associated trajectories $\mathcal{I}_\tau$ have the same length and the same initial state, we obtain the corresponding multistep Bellman mapping component for that state, as $t \to \infty$. There are several ways to make this statement more precise and use it in the analysis. It is convenient to use the Markov chain property of $\{\mathcal{I}_t\}$ for this purpose.

The state trajectories $\{\mathcal{I}_t\}$ form a time-homogeneous finite-state Markov chain. Therefore, the empirical distribution of $\{\mathcal{I}_t\}$ converges to some invariant distribution of this chain. To simplify exposition, let us assume, without loss of generality, that the chain has a unique invariant distribution. We denote this distribution by $\mathbb{P}_e$ and denote expectation under $\mathbb{P}_e$ by $\mathbb{E}_e$.

For every state i, $\frac{G_{t,i}(x)}{t}$ is the sample mean of the random affine functions

$$\delta\{i = (\mathcal{I}_\tau)_0\} \cdot \alpha_{i|\mathcal{I}_\tau|} \cdot \left(x_i - \tilde{T}_{\mathcal{I}_\tau} x\right), \qquad \tau = 1, 2, \ldots, t.$$

(In matrix representation they are given by $f_1(\mathcal{I}_\tau) + f_2(\mathcal{I}_\tau)\, x$ for some real-valued function f_1 and $\Re^n$-valued function f_2 on the space of trajectories.) So as t goes to infinity, $\frac{G_{t,i}(x)}{t}$ converges to the "mean" affine function

$$\mathbb{E}_e\Big[\delta\{i = (\mathcal{I})_0\} \cdot \alpha_{i|\mathcal{I}|} \cdot \left(x_i - \tilde{T}_{\mathcal{I}} x\right)\Big],$$

where the expectation is over $\mathcal{I}$, a random state sequence with distribution $\mathbb{P}_e$. By calculating this "mean" function, we obtain an expression of the projected Bellman equation associated with Algorithm I as follows.

Define a function $\bar{w}(\mathcal{I})$ for scaling purposes: for $\mathcal{I} = (i_0, \ldots, i_\ell)$, $\bar{w}(\mathcal{I}) = \alpha_{i_0 \ell}$. This is the weight of the new function term that would be included in G_t at step 2 if $\mathcal{I}_t = \mathcal{I}$.

Proposition 3.1. *Let $\mathcal{I}$ be a random sequence distributed as $\mathbb{P}_e$ and suppose that $\bar{w} = \mathbb{E}_e[\bar{w}(\mathcal{I})] > 0$. Then the sequences of functions, $\{f_{v,t}\}, v \in V_S$, in Algorithm I converge to:*

$$f_{v,t}(x) \to \bar{w} \cdot \left\langle v\,,\, x - T^{(\mathbf{c})} x \right\rangle_\xi, \qquad \forall\, v \in V_S,$$

where the vector ξ is given by

$$\xi_i \,\propto\, \mathbb{P}_e\big((\mathcal{I})_0 = i\big) \cdot \mathbb{E}_e\big[\,\alpha_{i|\mathcal{I}|} \mid (\mathcal{I})_0 = i\,\big], \qquad i = 1, \ldots, n,$$

and the collection $\mathbf{c}$ *of coefficients are given by: for each* i, $c_{ik} = 0$ *for* $k > N$, *and*

$$c_{ik} \propto \mathbb{P}_e\left(|\mathcal{I}| = k \,\middle|\, (\mathcal{I})_0 = i\right) \cdot \alpha_{ik}, \qquad 1 \le k \le N.$$

If $(\xi, \mathbf{c}, S)$ *satisfies the Regularity Condition, then the sequence of solutions of Equation*(t), $t \ge 1$, *in Algorithm I converges to the solution of the projected Bellman equation* $x = \Pi_\xi T^{(\mathbf{c})} x$.

Proof. Based on the way the state trajectories are generated, $\mathbb{P}_e$ has the following product form: for any state sequence $(i_0, \ldots, i_\ell)$, $\ell \le N$,

$$\mathbb{P}_e\big(\mathcal{I} = (i_0, \ldots, i_\ell)\big) = \mathbb{P}_e\big((\mathcal{I})_0 = i_0\big) \cdot \mathbb{P}_e\big(|\mathcal{I}| = \ell \,\big|\, (\mathcal{I})_0 = i_0\big) \cdot \textstyle\prod_{j=0}^{\ell-1} \bar{p}_{i_j i_{j+1}}. \tag{3.3}$$

This implies that for every state i and $k \le N$,

$$\mathbb{E}_e\big[\tilde{T}_{\mathcal{I}}\, x \,\big|\, (\mathcal{I})_0 = i, |\mathcal{I}| = k\big] = T_i^k\, x.$$

Therefore, for $i = 1, \ldots, n$,

$$\begin{aligned} \lim_{t\to\infty} \tfrac{G_{t,i}(x)}{t} &= \mathbb{E}_e\Big[\delta\{i = (\mathcal{I})_0\} \cdot \alpha_{i|\mathcal{I}|} \cdot \big(x_i - \tilde{T}_{\mathcal{I}} x\big)\Big] \\ &= \mathbb{P}_e\big((\mathcal{I})_0 = i\big) \cdot \textstyle\sum_{k=1}^N \mathbb{P}_e\big(|\mathcal{I}| = k \,\big|\, (\mathcal{I})_0 = i\big) \cdot \alpha_{ik} \cdot \big(\, x_i - T_i^k\, x\,\big) \qquad (3.4) \\ &= \bar{w} \cdot \xi_i \cdot \big(x_i - T_i^{(\mathbf{c})} x\big), \end{aligned}$$

where the last expression is obtained from Eq. (3.4) by normalization of the weights. In particular, ξ_i is proportional to the total weight of the terms $(x_i - T_i^k\, x)$, $1 \le k \le N$, and for each i, c_{ik} is proportional to the weight of the term $-T_i^k x$. Their values can be read off from Eq. (3.4) and are as stated in the proposition. This completes the proof. □

A similar analysis gives the projected weighted Bellman equation associated with Variant B, stated below. Here, for scaling purposes, we define the function $\bar{w}(\mathcal{I})$ by: for $\mathcal{I} = (i_0, \ldots, i_\ell)$,

$$\bar{w}(\mathcal{I}) = \textstyle\sum_{j=0}^{\ell - N_0} \delta\big\{i_j \in \mathrm{List}_{i_0}\big\} \cdot \alpha_{i_j(\ell - j)}.$$

This is the total weight of the new function terms that would be added at step $2''$ if $\mathcal{I}_t = \mathcal{I}$.

Proposition 3.2. *Let* $\mathcal{I}$ *be a random sequence distributed as* $\mathbb{P}_e$ *and suppose that* $\bar{w} = \mathbb{E}_e[\bar{w}(\mathcal{I})] > 0$. *Then the conclusions of Prop. 3.1 hold for Variant B of Algorithm I, with the vector* ξ *and coefficients* $\mathbf{c}$ *given by:*

$$\xi_i \propto \textstyle\sum_{k=N_0}^N \mathbb{P}_e\left((\mathcal{I})_{|\mathcal{I}|-k} = i,\ i \in \mathrm{List}_{(\mathcal{I})_0}\right) \cdot \alpha_{ik}, \qquad i = 1, \ldots, n,$$

and for each i, $c_{ik} = 0$ *for* $k > N$ *or* $k < N_0$, *and*

$$c_{ik} \propto \mathbb{P}_e\left((\mathcal{I})_{|\mathcal{I}|-k} = i,\ i \in \mathrm{List}_{(\mathcal{I})_0}\right) \cdot \alpha_{ik}, \qquad N_0 \le k \le N.$$

Proof. By Eq. (3.3), for every state i and every pair of positive integers ℓ and j with $j < \ell \le N$, and every sequence $(i_0, \ldots, i_{j-1})$, we have

$$\mathbb{E}_e\Big[\, \tilde{T}_{(\mathcal{I})_j^\ell}\, x \;\Big|\; (\mathcal{I})_0^j = (i_0, \ldots, i_{j-1}, i),\ |\mathcal{I}| = \ell \,\Big] = T_i^{\ell - j}\, x.$$

Direct calculation then shows that for every state i,

$$\begin{aligned} \lim_{t\to\infty} \tfrac{G_{t,i}(x)}{t} &= \mathbb{E}_e\Big[\, \textstyle\sum_{j=0}^{|\mathcal{I}|-N_0} \delta\big\{i = (\mathcal{I})_j,\, i \in \mathrm{List}_{(\mathcal{I})_0}\big\} \cdot \alpha_{i(|\mathcal{I}|-j)} \cdot \Big(x_i - \tilde{T}_{(\mathcal{I})_j^{|\mathcal{I}|}} x\Big)\,\Big] \\ &= \textstyle\sum_{k=N_0}^N \mathbb{P}_e\Big((\mathcal{I})_{|\mathcal{I}|-k} = i,\, i \in \mathrm{List}_{(\mathcal{I})_0}\Big) \cdot \alpha_{ik} \cdot \big(x_i - T_i^k\, x\big), \end{aligned}$$

where we define $(\mathcal{I})_j = \emptyset$ if $j < 0$. Similar to the proof of Prop. 3.1, from the above expression, we can then read off the weights ξ and coefficients $\mathbf{c}$, which make up the projected equation $x = \Pi_\xi T^{(\mathbf{c})} x$ obtained with the algorithm in the limit. □

3.3 A Block TD(λ)-Type Algorithm

We now introduce algorithms of the TD-type and show their connection to weighted Bellman mappings with state-dependent weights. We present a simpler algorithm first in this subsection, referred to as Algorithm II. It uses a single λ parameter for all states; it is similar to the TD(λ) method but uses only trajectories of finite length. We characterize its associated projected Bellman equation (Section 3.3.2) and compare the algorithm with the usual TD methods (Section 3.3.3). We will later give two variant algorithms that use state-dependent λ parameters (Section 3.4).

Like the algorithms discussed earlier, Algorithm II generates a state trajectory at each iteration t, forms multiple function terms based on the trajectory, and includes them in the function $G_t(x)$ to define the tth equation. The computation is efficient: each iteration is broken down into sub-iterations corresponding to each state transition along the trajectory. The algorithm maintains an equation at each sub-iteration based on the trajectory generated so far, and when a new transition is generated, it can update the equation with little work. A precise description of an iteration is given below. Its computation efficiency will be better seen from the recursive formulas given in the sequel. Let $\lambda \in [0,1]$. (The case of interest is $\lambda \in (0,1)$.)

Algorithm II: (iteration t)

(1) Choose a starting state i_0 and an integer ℓ with $1 \leq \ell \leq N$ as in Algorithm I.

(2) Do ℓ sub-iterations, starting from $k = 1, 2, \ldots$, as follows. At sub-iteration k:

(a) Generate a transition (i_{k-1}, i_k) and let $\mathcal{I} = (i_0, \ldots, i_k)$ be the sequence obtained so far.

(b) Define $Y_k \in \mathcal{A}_n[x]$ by: for $i = 1, \ldots, n$,

$$Y_{k,i}(x) = \sum_{j=0}^{k-1} \delta\{i = i_j\} \cdot \left((1-\lambda) \sum_{m=j+1}^{k-1} \lambda^{m-1-j} \Big(x_i - \tilde{T}_{(\mathcal{I})_j^m} x \Big) + \lambda^{k-1-j} \Big(x_i - \tilde{T}_{(\mathcal{I})_j^k} x \Big) \right).$$

(c) Form a system of equations:

$$x \in S, \qquad \tfrac{1}{t} \left\langle v, G_{t-1}(x) + Y_k(x) \right\rangle = 0, \quad \forall\, v \in V_S. \qquad \text{(Equation}(t;k)\text{)}$$

(3) Let $\mathcal{I}_t = \mathcal{I}$ and $G_t(x) = G_{t-1}(x) + Y_\ell(x)$.

(4) Form a system of equations:

$$x \in S, \qquad f_{v,t}(x) = 0, \quad \forall\, v \in V_S, \qquad \text{(Equation}(t)\text{)}$$

where $f_{v,t}(x) = \left\langle v, \frac{G_t(x)}{t} \right\rangle$.

The function in step 2b can be alternatively written as

$$Y_{k,i}(x) = \sum_{j=0}^{k-1} \delta\{i = i_j\} \cdot \sum_{m=j+1}^{k} \lambda^{m-1-j} \cdot \Big(\tilde{T}_{(\mathcal{I})_j^{m-1}} x - \tilde{T}_{(\mathcal{I})_j^m} x \Big),$$

where we define $\tilde{T}_{(\mathcal{I})_j^j}$ to be the mapping $\tilde{T}_{(\mathcal{I})_j^j} x = x_{i_j}$ for all x. The terms $(\tilde{T}_{(\mathcal{I})_j^{m-1}} x - \tilde{T}_{(\mathcal{I})_j^m} x)$ correspond to the so-called "temporal differences," which will appear in the recursive computation formulas of the algorithm.

3.3.1 Recursive Computation

Algorithm II can be implemented efficiently with recursive computation, like in standard TD methods. There is no need to store the trajectory during an iteration. We give the recursive formulas

below (their derivation is given in Appendix B.1). Suppose $V_S = \{v_1, \ldots, v_d\}$. Let Φ be the $n \times d$ matrix $(v_1 \ \ldots \ v_d)$ and let $\phi(i)'$ be its ith row, where $'$ denotes transpose, i.e.,

$$\phi(i) = \big(v_{1,i}, \ldots, v_{d,i}\big)'.$$

Equation(t) is stored after the change of variable $x = \Phi r$, where $r \in \Re^d$, as

$$f_{v_j,t}(\Phi r) = 0, \qquad j = 1, \ldots, d,$$

or in matrix representation,

$$\text{Equation}(t): \qquad C_t r - b_t = 0,$$

where C_t is a $d \times d$ matrix and b_t is a d-dimensional vector. Similarly, Equation$(t;k)$ at sub-iteration k is stored after the change of variable $x = \Phi r$ as

$$\text{Equation}(t;k): \qquad C_{t,k} r - b_{t,k} = 0,$$

where $C_{t,k}$ is a $d \times d$ matrix and $b_{t,k}$ is a d-dimensional vector. These matrices and vectors are computed recursively, by using d-dimensional auxiliary vectors Z_k as follows.

Recursive Formulas of Algorithm II

- At the beginning of iteration t, let

$$Z_0 = 0, \qquad C_{t,0} = \big(1 - \tfrac{1}{t}\big) C_{t-1}, \qquad b_{t,0} = \big(1 - \tfrac{1}{t}\big) b_{t-1}.$$

- For $k = 0, \ldots, \ell - 1$, let

$$\begin{aligned} Z_{k+1} &= \lambda\beta\rho_{k-1,k} \cdot Z_k + \phi(i_k), \\ C_{t,k+1} &= C_{t,k} + \tfrac{1}{t}\, Z_{k+1} \cdot \big(\phi(i_k) - \beta\rho_{k,k+1}\, \phi(i_{k+1})\big)', \\ b_{t,k+1} &= b_{t,k} + \tfrac{1}{t}\, Z_{k+1} \cdot \rho_{k,k+1}\, g_\mu(i_k, i_{k+1}), \end{aligned}$$

  where $\rho_{j,j+1} = 1$ for the case of ordinary PE [where $\tilde{T}_{\mathcal{I}}$ is defined by Eq. (3.1)], and $\rho_{j,j+1} = \frac{p_{i_j i_{j+1}}}{\bar{p}_{i_j i_{j+1}}}$ for the case of exploration-enhanced PE [where $\tilde{T}_{\mathcal{I}}$ is defined by Eq. (3.2)].

- At the end of iteration t, let

$$C_t = C_{t,\ell}, \qquad b_t = b_{t,\ell}.$$

From the above formulas, it can be seen that Algorithm II is similar to the LSTD(λ) algorithm, a least squares-based TD(λ)-type algorithm [Boy99, NB03, BY09, Yu10a], except that when one iteration ends and another starts, the vector Z_k is reset and subsequent updates start with $Z_0 = 0$. This helps control estimation variance (see Section 3.3.3).

3.3.2 Projected Weighted Bellman Equation in the Limit

We characterize the projected weighted Bellman equation obtained with Algorithm II in the limit. The analysis is similar to that for Algorithm I. We consider the affine function $Y_\ell(x)$, which is added to G_{t-1} at step 3 of iteration t, and we calculate its expectation (over a random sequence $\mathcal{I}$) under the invariant distribution $\mathbb{P}_e$ of the Markov chain $\{\mathcal{I}_t\}$. From this "mean" affine function, we then read off the weights ξ and the weighted Bellman mapping $T^{(\mathbf{c})}$, which, under the Regularity Condition, form a well-defined projected equation.

In what follows, for a state sequence $\mathcal{I}$, let $(\mathcal{I})_j = \emptyset$ for $j < 0$. Also, let the segment $(\mathcal{I})_0^j$ be $\emptyset$ if $j < 0$.

Proposition 3.3. *Let $\mathcal{I}$ be a random sequence distributed as $\mathbb{P}_e$ and let $\bar{w} = \mathbb{E}_e\big[|\mathcal{I}|\big]$. Then the sequences of functions, $\{f_{v,t}\}, v \in V_s$, in Algorithm II converge to:*

$$f_{v,t}(x) \to \bar{w} \cdot \big\langle v\,,\, x - T^{(\mathbf{c})}x \big\rangle_{\xi}, \qquad \forall\, v \in V_S,$$

where the vector ξ is given by

$$\xi_i \propto\ \mathbb{E}_e\big[\,\mathcal{C}_i(\mathcal{I})\,\big], \qquad i = 1, \ldots, n,$$

with $\mathcal{C}_i(\mathcal{I})$ being the number of times that state i appears in the segment $(\mathcal{I})_0^{|\mathcal{I}|-1}$; and the collection $\mathbf{c}$ of coefficients are given by: for each i, $c_{ik} = 0$ for $k > N$, and

$$c_{ik} \propto\ (1-\lambda)\lambda^{k-1} \cdot \mathbb{E}_e\big[\,\mathcal{C}_{ik}(\mathcal{I})\big] + \lambda^{k-1} \cdot \mathbb{P}_e\big((\mathcal{I})_{|\mathcal{I}|-k} = i\big), \qquad 1 \le k \le N,$$

where $\mathcal{C}_{ik}(\mathcal{I})$ is the number of times that state i appears in the segment $(\mathcal{I})_0^{|\mathcal{I}|-1-k}$. Moreover, if $(\xi, \mathbf{c}, S)$ satisfies the Regularity Condition, then the sequence of solutions of Equation(t), $t \ge 1$, in Algorithm II converges to the solution of the projected Bellman equation $x = \Pi_\xi\, T^{(\mathbf{c})}x$.

Proof. The ith component of $Y_\ell(x)$ is given by

$$\begin{aligned} Y_{\ell,i}(x) &= \sum_{j=0}^{\ell-1} \delta\big\{i = i_j\big\} \cdot \left((1-\lambda) \sum_{m=j+1}^{\ell-1} \lambda^{m-1-j} \Big(x_i - \tilde{T}_{(\mathcal{I}_t)_j^m}\, x \Big) + \lambda^{\ell-1-j} \Big(x_i - \tilde{T}_{(\mathcal{I}_t)_j^\ell}\, x \Big) \right) \\ &= \sum_{j=0}^{\ell-1} \delta\big\{i = i_j\big\} \cdot \left((1-\lambda) \sum_{k=1}^{\ell-1-j} \lambda^{k-1} \Big(x_i - \tilde{T}_{(\mathcal{I}_t)_j^{j+k}}\, x \Big) + \lambda^{\ell-1-j} \Big(x_i - \tilde{T}_{(\mathcal{I}_t)_j^\ell}\, x \Big) \right). \end{aligned} \tag{3.5}$$

For any $i, \bar{i}_0 \in \{1, \ldots, n\}$ and positive integer $\ell \le N$, denote

$$q(x; i, \bar{i}_0, \ell) = \mathbb{E}_e\Big[\, Y_{\ell,i}(x) \,\Big|\, (\mathcal{I})_0 = \bar{i}_0,\ |\mathcal{I}| = \ell \,\Big],$$

and using Eq. (3.3) and the expression of $Y_{\ell,i}(x)$ given in Eq. (3.5), we obtain

$$\begin{aligned} q(x; i, \bar{i}_0, \ell) = \sum_{j=0}^{\ell-1} & \mathbb{E}_e\Big[\, \delta\big\{i = (\mathcal{I})_j\big\} \,\Big|\, (\mathcal{I})_0 = \bar{i}_0,\ |\mathcal{I}| = \ell \,\Big] \\ & \cdot \left((1-\lambda) \sum_{k=1}^{\ell-1-j} \lambda^{k-1} \Big(x_i - T_i^k\, x \Big) + \lambda^{\ell-1-j} \Big(x_i - T_i^{\ell-j}\, x \Big) \right). \end{aligned} \tag{3.6}$$

The above expression is equal to

$$\begin{aligned} & \sum_{k=1}^{N} \mathbb{E}_e \left[\sum_{j=0}^{\ell-1} \delta\big\{i = (\mathcal{I})_j,\ j \le \ell - 1 - k\big\} \,\Big|\, (\mathcal{I})_0 = \bar{i}_0,\ |\mathcal{I}| = \ell \right] \cdot (1-\lambda)\lambda^{k-1} \cdot \Big(x_i - T_i^k\, x \Big) \\ & + \sum_{k=1}^{N} \mathbb{E}_e \Big[\, \delta\big\{i = (\mathcal{I})_{\ell-k}\big\} \,\Big|\, (\mathcal{I})_0 = \bar{i}_0,\ |\mathcal{I}| = \ell \,\Big] \cdot \lambda^{k-1} \cdot \Big(x_i - T_i^k\, x \Big). \end{aligned}$$

By taking the expectation over the initial state and length of $\mathcal{I}$, we obtain

$$\mathbb{E}_e\,\big[\, Y_{|\mathcal{I}|,i}(x)\big] = \sum_{k=1}^{N} \Big((1-\lambda)\lambda^{k-1} \cdot \mathbb{E}_e\big[\,\mathcal{C}_{ik}(\mathcal{I})\,\big] + \lambda^{k-1} \cdot \mathbb{P}_e\big((\mathcal{I})_{|\mathcal{I}|-k} = i\big) \Big) \cdot \Big(x_i - T_i^k\, x \Big), \tag{3.7}$$

and this is the function $\lim_{t\to\infty} \frac{G_{t,i}(x)}{t}$. We can then write

$$\lim_{t\to\infty} \frac{G_{t,i}(x)}{t} = \bar{w} \cdot \xi_i \cdot \big(x_i - T_i^{(\mathbf{c})} x\big),$$

where the coefficients $\mathbf{c}$ can be read off from Eq. (3.7) and are as stated in the proposition. The weights ξ_i are proportional to the total weight of the terms $(x_i - T_i^k\, x), 1 \le k \le N$, so from Eq. (3.6) we see that they are given by

$$\xi_i \propto \mathbb{E}_e\Big[\, \textstyle\sum_{j=0}^{|\mathcal{I}|-1} \delta\{i = (\mathcal{I})_j\}\, \Big] = \mathbb{E}_e\big[\mathcal{C}_i(\mathcal{I})\big].$$

Thus $\{f_{v,t}(x)\}$ converges to $\bar{w} \cdot \big\langle v\,,\, x - T^{(\mathbf{c})} x \big\rangle_\xi$ for all $v \in V_S$. The proposition then follows. □

As a special case of the preceding proposition, suppose the trajectories are consecutive blocks of N transitions from a single trajectory of a Markov chain $\{i_\tau\}$ with transition probabilities $\{\bar{p}_{ij}\}$. Suppose also that this chain has no periodic states. Then ξ is some invariant distribution of this Markov chain, and for each state i, $c_{ik} = 0$ for $k > N$, and

$$c_{ik} \propto (1-\lambda)\lambda^{k-1}\,(N-k) \,+\, \lambda^{k-1}, \qquad 1 \le k \le N. \tag{3.8}$$

(So in this case the sequence $\{c_{ik}\}$ is the same for all states i.) Figure 4 (left) plots the above $\{c_{ik}\}$ for $N = 100, \lambda = 0.99$, and compares them with the coefficients $c_{ik} = (1-\lambda)\lambda^{k-1}$ in the $T^{(\lambda)}$ mappings for $\lambda = 0.99, 0.97$, which are associated with the standard TD(λ) method.

3.3.3 Comparison with Usual TD(λ) Methods

Algorithm II can be especially useful in exploration-enhanced PE. It is well-known that using multistep Bellman mappings can decrease the approximation bias in the projected equation approach, but it tends to increase the estimation variance, partly due to the high variances of long-term cost estimates. The variance issue becomes acute especially in exploration-enhanced PE, where the computation involves products of importance-sampling weights (ratios between transition probabilities) along state trajectories. In that case, for example, for a usual, non-block TD(λ)-type algorithm, LSTD(λ) with $\lambda > 0$, even infinite asymptotic estimation variance can occur, despite the almost sure convergence of the algorithm [Yu10a, Yu10b]. By contrast, this cannot happen with the block-type Algorithm II, because it uses trajectories of no more than N transitions at each iteration.

Figure 4 (right) demonstrates this difference using a two-state example from [Yu10b, Appendix], for which analysis and empirical evidence suggest that the estimates of the (least squares) TD(0.99) converge at a rate slower than $1/\sqrt{t}$. We run Algorithm II as described in the special case given just before this subsection, with $N = 100$ and $\lambda = 0.99$. (The discount factor is 0.99 and S is one-dimensional.) The two algorithms obtain different projected equations in the limit. Plotted in Fig. 4 (right) are their estimates of the constant term of the associated equation (in low-dimensional representations). The dotted horizontal lines show the limits of the corresponding estimates. We plotted the estimates down-sampled at uneven time intervals with the X-axis indicating $\sqrt{t}$, to show that the two algorithms are converging at speeds of different orders.

3.4 TD Variants with State-Dependent λ Parameters

We introduce two variants of Algorithm II with state-dependent parameters $\lambda_i \in [0,1], i = 1, \ldots, n$.

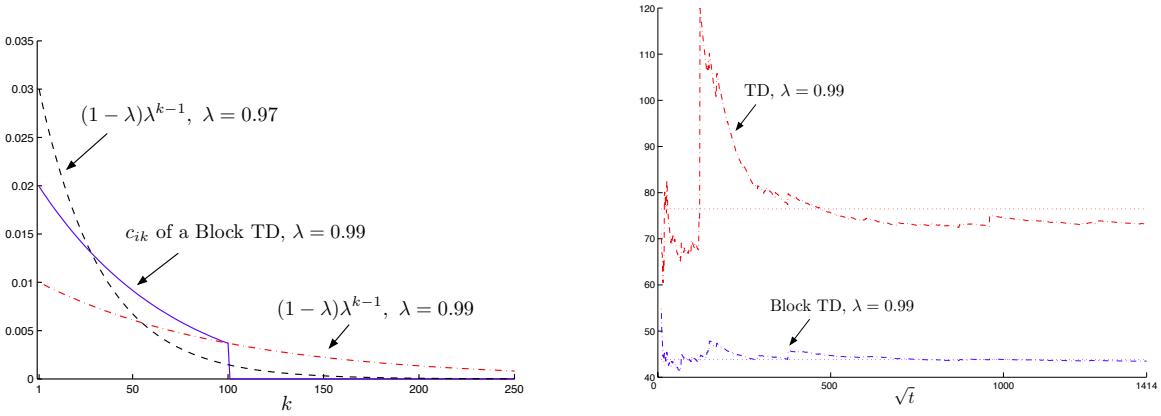

Figure 4: Differences between the TD(λ) method and the block TD(λ) method, illustrated with an example. Details are given in Sections 3.3.2 and 3.3.3.

3.4.1 Variant A

We replace λ by λ_i in the function $Y_{k,i}(x)$ at step 2b of Algorithm II:

(2b') Define $Y_k \in \mathcal{A}_n[x]$ by: for $i = 1, \ldots, n$,

$$Y_{k,i}(x) = \sum_{j=0}^{k-1} \delta\{i = i_j\} \cdot \left((1-\lambda_i) \sum_{m=j+1}^{k-1} \lambda_i^{m-1-j} \left(x_i - \tilde{T}_{(\mathcal{I})_j^m} x \right) + \lambda_i^{k-1-j} \left(x_i - \tilde{T}_{(\mathcal{I})_j^k} x \right) \right).$$

For the resulting algorithm, referred to as Variant A, we can characterize its associated projected Bellman equation $x = \Pi_\xi T^{(\mathbf{c})} x$ as follows. The derivation is similar to that for Algorithm II.

Proposition 3.4. *Assume the conditions of Prop. 3.3. Then the conclusions of Prop. 3.3 hold for Variant A, with λ_i in place of λ in the expressions of the coefficients c_{ik} for each state i.*

Variant A can be implemented efficiently, when $\{\lambda_i\}$ take a relatively small number of distinct values $\{\bar{\lambda}_1, \ldots, \bar{\lambda}_s\}$. The recursive formulas, given below, are the same as those of Algorithm II (cf. Section 3.3.1), except that we define the auxiliary vectors Z_k differently and we introduce s more d-dimensional auxiliary vectors $Z_k^{(1)}, \ldots, Z_k^{(s)}$. (See Appendix B.2 for the derivation.)

Recursive Formulas of Variant A

- At the beginning of iteration t, let $Z_0^{(m)} = 0$, $m = 1, \ldots, s$, and let

$$Z_0 = 0, \qquad C_{t,0} = \big(1 - \tfrac{1}{t}\big) C_{t-1}, \qquad b_{t,0} = \big(1 - \tfrac{1}{t}\big) b_{t-1}.$$

- For $k = 0, \ldots, \ell - 1$, let

$$Z_{k+1}^{(m)} = \bar{\lambda}_m \cdot \beta \rho_{k-1,k} \cdot Z_k^{(m)} + \delta\{\lambda_{i_k} = \bar{\lambda}_m\} \cdot \phi(i_k), \tag{3.9}$$

$$Z_{k+1} = \textstyle\sum_{m=1}^{s} Z_{k+1}^{(m)}, \tag{3.10}$$

and let

$$C_{t,k+1} = C_{t,k} + \tfrac{1}{t} Z_{k+1} \cdot \big(\phi(i_k) - \beta \rho_{k,k+1} \phi(i_{k+1})\big)',$$
$$b_{t,k+1} = b_{t,k} + \tfrac{1}{t} Z_{k+1} \cdot \rho_{k,k+1} g_\mu(i_k, i_{k+1}).$$

- At the end of iteration t, let $C_t = C_{t,\ell}, b_t = b_{t,\ell}$.

A Non-Block Version

With these formulas, we also obtain a non-block TD-type algorithm which employs the weighted Bellman mapping given by Eqs. (2.4)-(2.5):

$$T_i^{(\mathbf{c})} = (1-\lambda_i)\sum_{k\geq 1}\lambda_i^{k-1}T_i^k, \quad \text{if } \lambda_i \in [0,1); \qquad T_i^{(\mathbf{c})}x \equiv x_i^*, \ \ \forall\, x, \quad \text{if } \lambda_i = 1,$$

thus generalizing the standard LSTD algorithm. In particular, consider the sub-iterations only and let k increase continuously. The recursive formulas (3.9)-(3.10) for Z_k, together with

$$\begin{aligned} C_{k+1} &= \big(1-\tfrac{1}{k}\big)\, C_k + \tfrac{1}{k}\, Z_{k+1}\cdot\big(\phi(i_k) - \beta\rho_{k,k+1}\,\phi(i_{k+1})\big)', \\ b_{k+1} &= \big(1-\tfrac{1}{k}\big)\, b_k + \tfrac{1}{k}\, Z_{k+1}\cdot\rho_{k,k+1}\, g_\mu(i_k, i_{k+1}), \end{aligned}$$

produce a sequence of equations,

$$C_k r - b_k = 0, \quad k = 1, 2, \ldots,$$

which converges to a low-dimensional representation of the projected weighted Bellman equation

$$x = \Pi_\xi T^{(\mathbf{c})}x,$$

where $T^{(\mathbf{c})}$ is given by Eqs. (2.4)-(2.5) and ξ is an invariant distribution of the Markov chain with transition probabilities $\{\bar{p}_{ij}\}$. This convergence, for both ordinary and exploration-enhanced PE, follows from the results of [Yu10a, Yu10b]. Because the algorithm is not block-type, the convergence arguments are quite different from what we used in this paper, so we do not include the proof here. (A proof can be found in Section 4.3 of the recent, revised version of [Yu10b].)

3.4.2 Variant B

At step 2b of Algorithm II, we replace λ by λ_i and replace $\delta\{i = i_j\}$ by $\delta\{i = i_j, \lambda_i = \lambda_{i_0}\}$ in the function $Y_{k,i}(x)$:

($2b''$) Define $Y_k \in \mathcal{A}_n[x]$ by: for $i = 1, \ldots, n$,

$$Y_{k,i}(x) = \sum_{j=0}^{k-1}\delta\big\{i = i_j\,,\, \lambda_i = \lambda_{i_0}\big\}\cdot\left((1-\lambda_i)\sum_{m=j+1}^{k-1}\lambda_i^{m-1-j}\Big(x_i - \tilde{T}_{(\mathcal{I})_j^m}x\Big) + \lambda_i^{k-1-j}\Big(x_i - \tilde{T}_{(\mathcal{I})_j^k}x\Big)\right).$$

In other words, we include only the function terms corresponding to those trajectory segments that start from states whose λ parameters equal λ_{i_0}. This variant does not use every segment of a trajectory, but requires less memory than Variant A. Its recursive formulas are the same as those of Algorithm II, except that we redefine Z_{k+1} for $k < \ell$ to be

$$Z_{k+1} = \lambda_{i_0}\cdot\beta\rho_{k-1,k}\cdot Z_k + \delta\{\lambda_{i_k} = \lambda_{i_0}\}\cdot\phi(i_k).$$

(See Appendix B.2 for the derivation.) The projected weighted Bellman equation associated with Variant B can also be characterized similarly; we omit the derivation.

4 Approximation by State Aggregation

If the approximation subspace is given by

$$S = \Big\{ x \,\Big|\, x_i = \sum_{j \in E} \nu_{ij} x_j,\ i = 1, \ldots, n \Big\}$$

for some subset E of states and scalars ν_{ij}, and we let ξ be such that $\{i \mid \xi_i > 0\} \supset E$, then the projection Π_ξ onto S with respect to the semi-norm $\|\cdot\|_\xi$ is well-defined [see the discussion after the condition (2.8)]. Among special cases of this are a class of state aggregation methods. In addition to using a semi-norm projection, however, the approximation architectures of these methods also ensure that the projected equation $x = \Pi_\xi T^{(\mathbf{c})} x$ has a unique solution, and hence the Regularity Condition is always satisfied. We explain this in detail below.

The state space is partitioned into subsets $E_0, E_1, \ldots, E_d$. On each of the sets $E_1, \ldots, E_d$, we approximate the costs by a constant. On E_0, we determine the approximation by using convex combinations of the approximate costs at states outside E_0. Correspondingly, the approximation subspace is

$$S = \Big\{ x \,\Big|\, x_i = x_j,\ \forall\, i, j, \in E_m,\ 1 \le m \le d;\ \ x_i = \sum_{j : j \notin E_0} \nu_{ij} x_j,\ \forall\, i \in E_0 \Big\} \tag{4.1}$$

for some nonnegative scalars $\nu_{ij}, i \in E_0, j \notin E_0$, with $\sum_{j : j \notin E_0} \nu_{ij} = 1$. The special case $E_0 = \emptyset$ is called hard aggregation. We see that Π_ξ is well-defined for all ξ such that

$$\{i \mid \xi_i > 0\} \cap E_m \neq \emptyset, \qquad m = 1, \ldots, d. \tag{4.2}$$

We now let ξ satisfy, besides (4.2),

$$\xi_i = 0, \quad \forall\, i \in E_0. \tag{4.3}$$

Consider a projected Bellman equation $x = \Pi_\xi T^{(\mathbf{c})} x$. If Φ is a matrix with columns $v_1, \ldots, v_d$ forming a basis of S, then since Π_ξ has the matrix representation

$$\Pi_\xi = \Phi D, \qquad \text{where } D = (\Phi' \Xi \Phi)^{-1} \Phi' \Xi,\ \Xi = diag(\xi),$$

by making the change of variable $x = \Phi r$, we see that the projected equation $x = \Pi_\xi T^{(\mathbf{c})} x$ has a unique solution if and only if the following equation has a unique solution:

$$r = D T^{(\mathbf{c})}(\Phi r), \qquad r \in \Re^d. \tag{4.4}$$

We now show that for a particular choice of Φ, Eq. (4.4) can be viewed as a Bellman equation for an artificial Markovian problem whose states are the sets $E_1, \ldots, E_d$. As a result, Eq. (4.4) and hence also the projected equation $x = \Pi_\xi T^{(\mathbf{c})} x$ has a unique solution.

In particular, choose a basis of S, $v_1, \ldots, v_d \in \Re^n$, as follows: for $m = 1, \ldots, d$,

$$v_m = \begin{pmatrix} v_{m,1} \\ \vdots \\ v_{m,n} \end{pmatrix} \qquad \text{with} \quad v_{m,i} = \begin{cases} 1 & \text{if } i \in E_m, \\ \sum_{j \in E_m} \nu_{ij} & \text{if } i \in E_0, \\ 0 & \text{otherwise,} \end{cases}$$

In terms of its rows, $\Phi = (v_1 \ \ldots \ v_d)$ is now the $n \times d$ matrix whose ith row is

$$(\underbrace{0 \ \ldots \ 0}_{m-1} \ 1 \ 0 \ \ldots) \qquad \text{if } i \in E_m, m \neq 0,$$

and

$$\Big(\sum_{j\in E_1}\nu_{ij} \quad \cdots \quad \sum_{j\in E_d}\nu_{ij}\Big) \qquad \text{if } i\in E_0.$$

Each row can be viewed as probabilities of transition from state i to artificial, aggregated states $E_1,\ldots,E_d$.

The $d\times n$ matrix $D=(\Phi'\Xi\Phi)^{-1}\Phi'\Xi$ has a similar interpretation. Using the definition of Φ and ξ [cf. Eqs. (4.2), (4.3)], a direct calculation shows that the mth row of D consists of zeros and $\xi_i, i\in E_m$, scaled as follows:

$$D_{mi}=\begin{cases}\frac{\xi_i}{\sum_{j\in E_m}\xi_j} & \text{if } i\in E_m,\\ 0 & \text{otherwise,}\end{cases} \qquad i=1,\ldots,n.$$

Each row of D can be viewed as probabilities of transition from the aggregated state E_m to states $1,\ldots,n$.

With these probabilistic interpretations of D and Φ, Eq. (4.4) can be viewed as a Bellman equation for a discounted semi-Markov process on the aggregated states $E_1,\ldots,E_d$. Starting from each of these states, the process evolves as it first "disaggregates" to some state in $\{1,\ldots,n\}$, then lingers among the states $\{1,\ldots,n\}$ for a while, and finally returns to an aggregated state (or gets absorbed at a terminal cost-free state). The dynamics of this process are described by the "disaggregation" probability matrix D, the mapping $T^{(\mathbf{c})}$, and the "aggregation" probability matrix Φ. Since the Bellman equation of a discounted semi-Markov process has a unique solution, it follows that Eq. (4.4) has a unique solution and hence so does the projected equation $x=\Pi_\xi T^{(\mathbf{c})}x$.

In the context of aggregation, often one specifies first the aggregation and disaggregation schemes, Φ and D, which in turn determine S and ξ. Usually one considers approximating the original MDP by the MDP on the aggregated states defined by D, T and Φ, involving the one-step Bellman mappings T for each policy. Hard aggregation is often applied, i.e., $E_0=\emptyset$ (see e.g., [Gor95, TV96]). The special case where each of the sets $E_1,\ldots,E_d$ consists of a single state is known as aggregation with representative states [BT96, Ber12] and is associated with calculating approximate costs at a grid of states and interpolating between these costs to obtain approximate costs for the remaining states. The case with $E_0\neq\emptyset$ and with multiple states aggregated into one state has been referred to as aggregation with representative features [Ber12].

As to learning or simulation-based APE algorithms for forming and solving the equation $x=\Pi_\xi T^{(\mathbf{c})}x$, there are a number of ways to design them. One may work directly with the associated Markovian problem $r=DT^{(\mathbf{c})}(\Phi r)$ on the aggregated states, generating trajectories according to the dynamics of the process described earlier (see [Ber12, Chap. 6.5]). Alternatively, one may apply the algorithms discussed in the earlier sections for forming projected equations $x=\Pi_\xi T^{(\mathbf{c})}x$. We discuss one example below.

Consider the variant A of Algorithm II given in Section 3.4.1, which is a TD-type algorithm with parameters $\lambda_i\in[0,1]$ for each state i. It can be applied in the state aggregation context with the following modification. At step $2b'$, we define Y_k as:

For $i=1,\ldots,n$,

$$Y_{k,i}(x)=\sum_{j=0}^{k-1}\delta\big\{i=i_j,\, i\notin E_0\big\}\cdot\left((1-\lambda_i)\sum_{m=j+1}^{k-1}\lambda_i^{m-1-j}\Big(x_i-\tilde{T}_{(\mathcal{I})_j^m}x\Big)+\lambda_i^{k-1-j}\Big(x_i-\tilde{T}_{(\mathcal{I})_j^k}x\Big)\right).$$

Correspondingly, the recursive formulas of Variant A are modified with a new definition of auxiliary

vectors Z_k and $Z_k^{(1)}, \ldots, Z_k^{(s)}$, assuming that $\{\bar{\lambda}_1, \ldots, \bar{\lambda}_s\}$ are the distinctive values of $\{\lambda_i\}$:

$$Z_{k+1}^{(m)} = \bar{\lambda}_m \cdot \beta \rho_{k-1,k} \cdot Z_k^{(m)} + \delta\{\lambda_{i_k} = \bar{\lambda}_m,\, i_k \notin E_0\} \cdot \phi(i_k), \qquad m = 1, \ldots, s, \tag{4.5}$$

$$Z_{k+1} = \textstyle\sum_{m=1}^{s} Z_{k+1}^{(m)}, \tag{4.6}$$

[cf. Eqs. (3.9) and (3.10)]. Here the vector ξ and equivalently the matrix D are determined by the algorithm. Similarly, one can also have a non-block version of the algorithm, like the one given in Section 3.4.1.

5 Applications in Approximate Policy Iteration

The exact policy iteration method generates a sequence of policies $\mu_k, k = 1, 2, \ldots$, by iterating between two steps: policy evaluation, which computes the cost vector J^{μ_k} of the policy μ_k, and policy improvement, which computes a policy μ_{k+1} with improved performance. One API approach imitates this procedure and evaluates the policies approximately; then the APE algorithms based on solving projected weighted Bellman equations can be directly applied. There are also API approaches that replace the "pure" policy evaluation step with something in between policy evaluation and value iteration. We focus on two such approaches in this section, of which one is based on modified policy iteration (see e.g., [Put94, Rot79, CR12]) and the idea of solving projected equations, and the other resembles asynchronous modified policy iteration (see e.g., [BT96, Ber12]). We demonstrate how weighted Bellman mappings and the algorithms discussed earlier can be applied within these API schemes.

5.1 Projected Equation-Based Methods

We describe first a general form of approximate modified policy iteration, which has a favorable performance bound as shown by Thiery and Scherrer [TS10]. Let $\{\nu_m\}$ be a nonnegative scalar sequence with $\sum_{m\geq 1} \nu_m = 1$. At the beginning of iteration k, we have from the previous iteration some cost vector J_{k-1} and a policy μ_k that is greedy with respect to J_{k-1}, which we obtained from the usual policy improvement step (i.e., μ_k is optimal for a two-stage problem with J_{k-1} as the terminal costs). To simplify notation, denote $\bar{\mu} = \mu_k, \bar{J} = J_{k-1}$, and $\nu = \{\nu_m\}$. Let

$$T_{\bar{\mu}}^{(\nu)} = \textstyle\sum_{m\geq 1} \nu_m T_{\bar{\mu}}^m,$$

where $T_{\bar{\mu}}$ is the Bellman mapping for $\bar{\mu}$. At iteration k, we let J_k be

$$J_k \approx T_{\bar{\mu}}^{(\nu)} \bar{J}. \tag{5.1}$$

Suppose that for every k, the approximation error at this step satisfies $\|J_k - T_{\bar{\mu}}^{(\nu)} \bar{J}\|_\infty \leq \epsilon$. Then, as established in [TS10], the performance of μ_k can be bounded asymptotically as

$$\limsup_{k\to\infty} \|J^{\mu_k} - J^*\|_\infty \leq \frac{2\beta}{(1-\beta)^2}\epsilon, \tag{5.2}$$

where J^* denotes the optimal cost vector.

Regarding approximation of the vector $T_{\bar{\mu}}^{(\nu)} \bar{J}$ in (5.1), one idea in [TS10] is to find an equation satisfied by $T_{\bar{\mu}}^{(\nu)} \bar{J}$ and then solve a projected version of the equation. Weighted Bellman mappings can be applied in this context and we give two examples below.

5.1.1 Application in λ-Policy Iteration

"λ-policy iteration" is introduced by Bertsekas and Ioffe [BI96] (see also [Ber11]). It corresponds to the case $\nu_m = (1-\lambda)\lambda^{m-1}$. The desired vector $T_{\bar{\mu}}^{(\nu)}\bar{J}$ in (5.1) is then

$$x^* = T_{\bar{\mu}}^{(\lambda)}\bar{J},$$

and an equation which has x^* as its unique solution is

$$x = Mx \stackrel{def}{=} (1-\lambda)T_{\bar{\mu}}\bar{J} + \lambda T_{\bar{\mu}}x.$$

To approximate x^*, we can solve a projected equation

$$x = \Pi_\xi M^{(\mathbf{c})}x \tag{5.3}$$

for certain ξ and $\mathbf{c}$. (This extends the proposal in [TS10], which is to solve $x = \Pi_\xi Mx$.) The algorithms given earlier for APE are applicable here. In particular, using trajectories of states, we can construct equations that asymptotically approach an equation of the form (5.3), and use their solutions as approximations of x^*.

The computation details are also similar to those in the APE case. Let $g_{\bar{\mu}}$ denote the expected one-stage cost vector of $\bar{\mu}$ and let $g_{\bar{\mu}}(i,i')$ denote the cost of transition from state i to i'. The equation $x = Mx$ is equivalent to the Bellman equation for the policy $\bar{\mu}$ in an MDP that has discount factor $\lambda\beta$ and has $\lambda g_{\bar{\mu}} + (1-\lambda)T_{\bar{\mu}}\bar{J}$ as the expected one-stage cost under $\bar{\mu}$. Therefore, the APE algorithms given in the earlier sections can be applied here with the discount factor being $\lambda\beta$ and with the cost of transition being $g_\mu(i_k, i_{k+1}) = g_{\bar{\mu}}(i_k, i_{k+1}) + (1-\lambda)\beta\bar{J}(i_{k+1})$ for a state transition (i_k, i_{k+1}) along a trajectory.

5.1.2 Application in Approximate Modified Policy Iteration

Consider the general approximate modified policy iteration scheme described at the beginning of Section 5.1, where we compute approximately $T_{\bar{u}}^{(\nu)}\bar{J}$ at the "policy evaluation" step (5.1). A modification of this scheme is to approximate the cost vector

$$x^* = (1-\gamma)T_{\bar{\mu}}^{(\nu)}\bar{J} + \gamma J^{\bar{\mu}}, \tag{5.4}$$

where $\gamma \in (0,1)$, instead of approximating $T_{\bar{\mu}}^{(\nu)}\bar{J}$. Using a large value of γ has the effect of taking a large step toward $J^{\bar{\mu}}$ when updating J_k and results in an API scheme that is close to policy iteration. We may let ν_m be positive for only a few small integers m, since for large m, $T_{\bar{\mu}}^m\bar{J} \approx J^{\bar{\mu}}$. The performance bound (5.2) [TS10] can be shown to hold for the API scheme we just proposed; moreover, this bound holds even if γ and ν are iteration-dependent.

To approximate x^*, we can apply the projected equation approach: first, we find a linear equation satisfied by x^*, and we then solve a projected version of that equation. For any $T_{\bar{\mu}}^{(\mathbf{c})}$, since

$$J^{\bar{\mu}} = T_{\bar{\mu}}^{(\mathbf{c})}J^{\bar{\mu}}$$

and $J^{\bar{\mu}} = \big(x^* - (1-\gamma)T_{\bar{\mu}}^{(\nu)}\bar{J}\big)/\gamma$, by substitution we see that x^* is the unique solution of the equation[9]

$$\zeta(x) = T_{\bar{\mu}}^{(\mathbf{c})}\zeta(x), \tag{5.5}$$

[9] Expressed as a fixed point equation in x, Eq. (5.5) is

$$x = (1-\gamma)\left(T_{\bar{\mu}}^{(\nu)}\bar{J} - \big(T_{\bar{\mu}}^{(\mathbf{c})} \circ T_{\bar{\mu}}^{(\nu)}\big)\bar{J}\right) + T_{\bar{\mu}}^{(\mathbf{c})}x.$$

where

$$\zeta(x) = \big(x - (1-\gamma)T_{\bar{\mu}}^{(\nu)}\bar{J}\big)/\gamma.$$

So to approximate x^*, we can solve a projected version of Eq. (5.5):

$$x \in S, \qquad \Pi_\xi\big(\zeta(x) - T_{\bar{\mu}}^{(\mathbf{c})}\zeta(x)\big) = 0. \tag{5.6}$$

As in the APE case, we need not fix ξ and $\mathbf{c}$ in advance; their values can be determined by the algorithms instead.

The APE algorithms we discussed in the earlier sections can be applied here. More specifically, in these algorithms, we now define the equation at iteration t to be

$$x \in S, \qquad \Big\langle v\,,\, \tfrac{G_t(\zeta(x))}{t}\Big\rangle = 0, \qquad \forall\, v \in V_S, \tag{5.7}$$

where we have $\zeta(x)$ in place of the variable x in the function $G_t(x)$. Regarding recursive computation, the formulas can be similarly derived as in the APE case, by making the change of variable $x = \Phi r$, and by using sample-based estimates for each component of the vector $(1-\gamma)T_{\bar{\mu}}^{(\nu)}\bar{J}$ in the function $\zeta(x)$, wherever these components are needed in the algorithms. The computation overhead for obtaining the estimates is not too high if only a few coefficients ν_m with small integers m are positive.

5.2 Asynchronous Modified Policy Iteration-Like Methods

Consider first a policy iteration algorithm without function approximation. At iteration k, given the cost vector $\bar{J} = J_{k-1}$ and a policy $\bar{\mu} = \mu_k$ that is greedy with respect to $\bar{J}$, we let

$$J_k = T_{\bar{\mu}}^{(\mathbf{c})}\bar{J} \tag{5.8}$$

for some weighted Bellman mapping $T_{\bar{\mu}}^{(\mathbf{c})}$ corresponding to $\bar{\mu}$. The coefficients $\mathbf{c}$ can be iteration-dependent.

This algorithm is a special case of asynchronous modified policy iteration. Monotonic convergence of $\{J_k\}$ to the optimal cost J^* is guaranteed if we start with a cost vector and policy pair (μ_0, J_0) such that $T_{\mu_0}J_0 \le J_0$ (this convergence follows from the monotonicity of $T_\mu^{(\mathbf{c})}$ for any μ). In general, without the initial condition, the algorithm differs significantly from "pure" or modified policy iteration methods in that the "policy improvement" steps do not yield improved policies. We expect its behavior to be similar to the general asynchronous modified policy iteration algorithm, which, despite its wide use in practice, is quite complex and not well-understood theoretically.

An approximate version of the above algorithm is obtained when we add projection in (5.8):

$$J_k = \Pi_\xi T_{\bar{\mu}}^{(\mathbf{c})}\bar{J}. \tag{5.9}$$

Computing the right-hand side of (5.9) is similar to solving a projected equation. The APE algorithms given earlier are applicable: we only need to replace all the terms $\tilde{T}_{\mathcal{I}}x$ in the equations with the terms $\tilde{T}_{\mathcal{I}}\bar{J}$. As before, we can let the algorithms determine the values of ξ and $\mathbf{c}$. Recursive formulas can also be similarly derived; we omit the details.

We note that the preceding API algorithm inherits the complex behavior of asynchronous modified policy iteration. While (5.9) may look similar to $J_k = \Pi_\xi T_{\bar{\mu}}^{(\lambda)}\bar{J}$ in λ-policy iteration or to $J_k \approx T_{\bar{\mu}}^{(\nu)}\bar{J}$ in approximate modified policy iteration, the performance bound (5.2) does not apply here in general.

Finally, we mention that there is another use of the approximate costs J_k computed by (5.9). They can be used as the stopping costs for the new, policy iteration-like Q-learning algorithms of [BY12, YB12], which solve sequences of optimal stopping problems where the stopping costs represent the current estimates of upper bounds of the optimal costs.

6 Discussion

In this paper we studied the use of weighted Bellman mappings for APE and API. This gave rise to a great variety of algorithms, which have the same approximation character as the widely applied, standard TD algorithms, and can be as easily implemented in practice. The important feature of our approximation framework is that one can choose different types of Bellman mappings for different states. One may use this as a means to balance approximation bias against estimation variance. By varying the mappings, one may also detect potential bias issues from the approximate solutions themselves, if they appear to be inconsistent with each other. There are still many open questions, however, when applying these mappings or Bellman mappings of even more general forms. For example, it is not clear how to choose the mapping in an "optimal" way, or how to even define the notion of optimality. Nevertheless, we believe that such mappings provide powerful and versatile means for approximations in MDP, and deserve further study and numerical experimentation.

References

[BB96] S. J. Bradtke and A. G. Barto, *Linear least-squares algorithms for temporal difference learning*, Machine Learning **22** (1996), 33–57.

[BBSE10] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst, *Reinforcement learning and dynamic programming using function approximators*, CRC Press, New York, 2010.

[Ber95] D. P. Bertsekas, *A counterexample to temporal differences learning*, Neural Computation **7** (1995), 270–279.

[Ber11] _____, *Lambda-policy iteration: A review and a new implementation*, Reinforcement Learning and Approximate Dynamic Programming for Feedback Control (F. Lewis and D. Liu, eds.), IEEE Press, 2011.

[Ber12] _____, *Dynamic programming and optimal control*, 4th ed., vol. II, Athena Scientific, Belmont, MA, 2012.

[BI96] D. P. Bertsekas and S. Ioffe, *Temporal differences-based policy iteration and applications in neuro-dynamic programming*, Tech. Report LIDS-P-2349, MIT, 1996.

[Boy99] J. A. Boyan, *Least-squares temporal difference learning*, Proc. the 16th Int. Conf. Machine Learning, 1999.

[BT96] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-dynamic programming*, Athena Scientific, Belmont, MA, 1996.

[BY09] D. P. Bertsekas and H. Yu, *Projected equation methods for approximate solution of large linear systems*, J. Computational and Applied Mathematics **227** (2009), no. 1, 27–50.

[BY12] _____, *Q-learning and enhanced policy iteration in discounted dynamic programming*, Mathematics of Operations Research **37** (2012), 66–94.

[CFHM07] H. S. Chang, M. C. Fu, J. Hu, and S. I. Marcus, *Simulation-based algorithms for Markov decision processes*, Springer, New York, 2007.

[CR12] P. G. Canbolat and U. G. Rothblum, *(Approximate) iterated successive approximations algorithm for sequential decision processes*, Annals of Operations research (2012), forthcoming.

[Gor95] G. J. Gordon, *Stable function approximation in dynamic programming*, Proc. the 12th Int. Conf. Machine Learning, 1995.

[Gos03] A. Gosavi, *Simulation-based optimization: Parametric optimization techniques and reinforcement learning*, Springer-Verlag, New York, 2003.

[KS00] M. Kearns and S. Singh, *Bias-variance error bounds for temporal difference updates*, COLT, 2000.

[KT03] V. R. Konda and J. N. Tsitsiklis, *Actor-critic algorithms*, SIAM J. Control and Optimization **42** (2003), 1143–1166.

[Mey08] S. Meyn, *Control techniques for complex networks*, Cambridge University Press, Cambdrige, UK, 2008.

[MS10] H. R. Maei and R. S. Sutton, *GQ(λ): A general gradient algorithm for temporal-difference prediction learning with eligibility traces*, Proc. the 3d Conf. on Artificial General Intelligence, 2010.

[MSB+09] H. R. Maei, C. Szepesvari, S. Bhatnagar, D. Silver, D. Precup, and R. S. Sutton, *Convergent temporal-difference learning with arbitrary smooth function approximation*, NIPS, 2009.

[NB03] A. Nedić and D. P. Bertsekas, *Least squares policy evaluation algorithms with linear function approximation*, Discrete Event Dynamic Systems: Theory and Applications **13** (2003), 79–110.

[Pow07] W. B. Powell, *Approximate dynamic programming: Solving the curses of dimensionality*, Wiley, New York, 2007.

[PSD01] D. Precup, R. S. Sutton, and S. Dasgupta, *Off-policy temporal-difference learning with function approximation*, Proc. the 18th Int. Conf. Machine Learning, 2001.

[Put94] M. L. Puterman, *Markov decision processes: Discrete stochastic dynamic programming*, John Wiley & Sons, New York, 1994.

[Rot79] U. G. Rothblum, *Iterated successive approximation for sequential decision processes*, Stochastic Control and Optimization (J. W. B. van Overhagen and H. C. Tijms, eds.), Vrije University, Amsterdam, 1979.

[Saa03] Y. Saad, *Iterative methods for sparse linear systems*, 2nd ed., SIAM, Philadelphia, 2003.

[SB98] R. S. Sutton and A. G. Barto, *Reinforcement learning*, MIT Press, Cambridge, MA, 1998.

[Sch10] B. Scherrer, *Should one compute the temporal difference fix point or minimize the Bellman residual? The unified oblique projection view*, Proc. the 27th Int. Conf. Machine Learning, 2010.

[Sut88] R. S. Sutton, *Learning to predict by the methods of temporal differences*, Machine Learning **3** (1988), 9–44.

[Sut95] ———, *TD models: Modeling the world at a mixture of time scales*, Proc. the 12th Int. Conf. Machine Learning, 1995, pp. 531–539.

[Sze10] C. Szepesvári, *Algorithms for reinforcement learning*, Morgan & Claypool, 2010.

[TS10] C. Thiery and B. Scherrer, *Least-squares λ policy iteration: Bias-variance trade-off in control problems*, Proc. the 27th Int. Conf. Machine Learning, 2010.

[TV96] J. N. Tsitsiklis and B. Van Roy, *Feature-based methods for large-scale dynamic programming*, Machine Learning **22** (1996), 59–94.

[TV97] ———, *An analysis of temporal-difference learning with function approximation*, EEE Transaction on Automatic Control **42** (1997), 674–690.

[UMKI11] T. Ueno, S. Maeda, M. Kawanabe, and S. Ishii, *Generalized TD learning*, Journal of Machine Learning Research **12** (2011), 1977–2020.

[YB10] H. Yu and D. P. Bertsekas, *Error bounds for approximations from projected linear equations*, Mathematics of Operations Research **35** (2010), 306–329.

[YB12] ———, *Q-learning and policy iteration algorithms for stochastic shortest path problems*, Annals of Operations Research (2012), forthcoming; DOI: 10.1007/s10479-012-1128-z.

[Yu10a] H. Yu, *Convergence of least squares temporal difference methods under general conditions*, Proc. the 27th Int. Conf. Machine Learning, 2010.

[Yu10b] ———, *Least squares temporal difference methods: An analysis under general conditions*, Technical Report C-2010-39, Dept. Computer Science, University of Helsinki, 2010, revised 2012; to appear in SIAM J. Control and Optimization.

Appendices

A Proofs for Section 2.2

In this appendix we verify some statements made in Section 2.2 about the approximation properties of the solution $\bar{x}$ of a projected Bellman equation $x = \Pi_\xi T^{(\mathbf{c})} x$. Recall that $T^{(\mathbf{c})}$ is a weighted Bellman mapping given by

$$T_i^{(\mathbf{c})} = \textstyle\sum_{k\geq 1} c_{ik} \cdot T_i^k \qquad i = 1, \ldots, n,$$

and it is denoted in matrix notation by $T^{(\mathbf{c})}x = Ax + b$. Recall also that $(\xi, \mathbf{c}, S)$ is assumed to satisfy the Regularity Condition (Section 2.1), and Π_ξ is the projection onto the approximation subspace S with respect to the semi-norm $\|\cdot\|_\xi$, given by

$$\Pi_\xi x = \arg\min_{y\in S} \|x - y\|_\xi, \qquad \forall\, x,$$

where the minimum is uniquely attained.

Let $\langle x, y\rangle_\xi = \sum_{i=1}^n \xi_i x_i y_i$. Some basic properties of Π_ξ and $\|\cdot\|_\xi$ are similar to those in the case where $\|\cdot\|_\xi$ is a norm:

(a) $\|x\|_\xi = 0$ implies $\Pi_\xi x = 0$;

(b) $\langle y, x - \Pi_\xi x\rangle_\xi = 0$ for any $y \in S$ (optimality condition);

(c) for any $y \in S$, $\|x - y\|_\xi^2 = \|x - \Pi_\xi x\|_\xi^2 + \|\Pi_\xi x - y\|_\xi^2$ (Pythagorean theorem);

(d) $\|\Pi_\xi x\|_\xi \leq \|x\|_\xi$ (nonexpansiveness);

(e) $\|x + y\|_\xi \leq \|x\|_\xi + \|y\|_\xi$ (triangle inequality).

From the definition of $\|L\|_\xi$ for an $n \times n$ matrix L, another property of the semi-norm $\|\cdot\|_\xi$ follows:

(f) $\|L\|_\xi < \infty$ if and only if $\|Ly\|_\xi = 0$ for all $y \in \Re^n$ with $\|y\|_\xi = 0$.

First, we show Prop. 2.3.

Proof of Prop. 2.3. We only need to verify the last statement of this proposition, which is about when the bound (2.12) is non-vacuous: if $\|\Pi_\xi A\|_\xi < \infty$, then

$$\tilde{B}(A, \xi, S) = \|(I - \Pi_\xi A)^{-1} \Pi_\xi A (I - \Pi_\xi)\|_\xi < \infty. \tag{A.1}$$

The special case where ξ is an invariant distribution of P is addressed by Lemma A.1 below.

Suppose $\|\Pi_\xi A\|_\xi < \infty$. Consider any $y \in \Re^n$ with $\|y\|_\xi = 0$. By property (a), $\Pi_\xi y = 0$, so $(I - \Pi_\xi)y = y$. By property (f), $\|\Pi_\xi A y\|_\xi = 0$, and since $\Pi_\xi A y \in S$, this implies $\Pi_\xi A y = 0$ by Eq. (2.8). Hence

$$(I - \Pi_\xi A)^{-1} \Pi_\xi A (I - \Pi_\xi) y = 0,$$

which implies, by property (f), Eq. (A.1). □

Next, we verify Cor. 2.1, which states that

$$\bar{x} = \Pi_\xi x^* \qquad \text{if } \|\Pi_\xi A\|_\xi < \infty \text{ and } \|x^* - \Pi_\xi x^*\|_\xi = 0.$$

Proof of Cor. 2.1. The assumption implies $\|\Pi_\xi A(x^* - \Pi_\xi x^*)\|_\xi = 0$ by property (f). Since the vector $\Pi_\xi A(x^* - \Pi_\xi x^*) \in S$, we have $\Pi_\xi A(x^* - \Pi_\xi x^*) = 0$ by Eq. (2.8), so

$$\Pi_\xi A(\Pi_\xi x^*) = \Pi_\xi A x^*.$$

Adding $\Pi_\xi b$ to both sides and using the fact $x^* = T^{(\mathbf{c})} x^*$, we obtain

$$\Pi_\xi T^{(\mathbf{c})}(\Pi_\xi x^*) = \Pi_\xi T^{(\mathbf{c})} x^* = \Pi_\xi x^*,$$

i.e., $\Pi_\xi x^*$ is the solution of the projected equation $x = \Pi_\xi T^{(\mathbf{c})} x$. Therefore, $\bar{x} = \Pi_\xi x^*$. □

The contraction-based bounds (2.13)-(2.15) require the condition

$$\|\Pi_\xi A\|_\xi < 1. \tag{A.2}$$

The bound (2.13) in Prop. 2.4 follows exactly from the arguments of Tsitsiklis and Van Roy [TV97], which use the Pythagorean theorem (property (c) above) in addition to (A.2).

In what follows, we consider the case where ξ is an invariant distribution of P: $\xi' P = \xi$, and we verify the bounds (2.14) and (2.15) in Cors. 2.2 and 2.3. (Of interest is the case where the Markov chain has transient states, so some components of ξ are zero.) First, we note that while with such ξ, Eq. (A.2) holds for standard TD(λ), this is not the case for weighted Bellman mappings involving state-dependent weights. However, it is true that $\|\Pi_\xi A\|_\xi < \infty$.

Lemma A.1. *Suppose ξ is an invariant distribution of P. Then,*

$$\|\Pi_\xi A\|_\xi \leq \|A\|_\xi < \infty.$$

Proof. The matrix A has entries

$$A_{ij} = \sum_{k\geq 1} c_{ik}\beta^k P_{ij}^k, \qquad i,j = 1,\ldots,n, \tag{A.3}$$

where P_{ij}^k denotes the (i,j)th entry of P^k. Since ξ is an invariant distribution of P, the states i with $\xi_i > 0$ form a closed set of the Markov chain. Consequently, for all $y \in \Re^n$ with $\|y\|_\xi = 0$, $\sum_{j=1}^n P_{ij}y_j = 0$ for all i with $\xi_i > 0$, which implies $\|Ay\|_\xi = 0$. So by property (f), $\|A\|_\xi < \infty$, and by property (d), $\|\Pi_\xi A\|_\xi \leq \|A\|_\xi < \infty$. □

It is true that $\|\Pi_\xi A\|_\xi < 1$ if the coefficients $\mathbf{c}$ in $T^{(\mathbf{c})}$ are state-independent: $c_{ik} = c_k$ for all states i. This and the more exact bound given in Eq. (2.15),

$$\|\Pi_\xi A\|_\xi \leq \sum_{k\geq 1} c_k\,\beta^k \leq \beta, \tag{A.4}$$

follow from the same arguments as in [TV97]. We include a proof.

Proof of Cor. 2.3. In this case, we have $A = \sum_{k\geq 1} c_k\,\beta^k P^k$. Since ξ is an invariant distribution of P, it follows that

$$\|P^k\|_\xi = \sup_{y:\|y\|_\xi=1} \|P^k y\|_\xi = 1, \qquad \forall\ k \geq 1. \tag{A.5}$$

(The supremum is attained at y with $y_i = 1$ for i with $\xi_i > 0$ and $y_i = 0$ for i with $\xi_i = 0$.) Then by the triangle inequality (property (e)),

$$\|A\|_\xi = \sup_{y:\|y\|_\xi=1} \Big\| \sum_{k\geq 1} c_k\,\beta^k P^k y \Big\|_\xi \leq \sum_{k\geq 1} c_k\,\beta^k \Big(\sup_{y:\|y\|_\xi=1} \|P^k y\|_\xi \Big) = \sum_{k\geq 1} c_k\,\beta^k.$$

By the nonexpansiveness of Π_ξ (property (d)), Eq. (A.4) follows. □

Finally, we verify the bound (2.14) in Cor. 2.2 for $T^{(\mathbf{c})}$ with state-dependent weights. Here it is assumed

$$\textstyle\sum_{k\geq 1} \bar{c}_k\beta^{2k} < 1, \qquad \text{where } \ \bar{c}_k = \max_{1\leq i\leq n} c_{ik},$$

and we need to show

$$\|\Pi_\xi A\|_\xi \leq \sqrt{\textstyle\sum_{k\geq 1} \bar{c}_k\beta^{2k}}. \tag{A.6}$$

Proof of Cor. 2.2. By Eq. (A.3), for any $y \in \Re^n$,

$$\|Ay\|_\xi^2 = \textstyle\sum_{i=1}^n \xi_i \left(\sum_{k\geq 1} c_{ik}\beta^k \sum_{j=1}^n P_{ij}^k y_j \right)^2.$$

For each term in the summation, using the convexity of the function x^2, the fact $\sum_{k\geq 1} c_{ik} = 1$ and the definition of $\bar{c}_k$, we have

$$\left(\textstyle\sum_{k\geq 1} c_{ik}\beta^k \sum_{j=1}^n P_{ij}^k y_j \right)^2 \leq \sum_{k\geq 1} c_{ik}\beta^{2k} \left(\sum_{j=1}^n P_{ij}^k y_j \right)^2 \leq \sum_{k\geq 1} \bar{c}_k\beta^{2k} \left(\sum_{j=1}^n P_{ij}^k y_j \right)^2.$$

Combining the preceding two relations with Eq. (A.5), we have

$$\|Ay\|_\xi^2 \leq \sum_{k\geq 1} \bar{c}_k\beta^{2k} \sum_{i=1}^n \xi_i \left(\sum_{j=1}^n P_{ij}^k y_j \right)^2 = \sum_{k\geq 1} \bar{c}_k\beta^{2k} \|P^k y\|_\xi^2 \leq \sum_{k\geq 1} \bar{c}_k\beta^{2k} \|y\|_\xi^2,$$

and therefore,

$$\|A\|_\xi \leq \sqrt{\textstyle\sum_{k\geq 1} \bar{c}_k\beta^{2k}}.$$

The bound (A.6) then follows by applying property (d). □

B Derivations of Recursive Formulas of TD-Type Algorithms

We verify in this appendix the recursive computation formulas of Algorithm II and its two variants. They were given in Section 3.3.1 and Section 3.4, respectively.

As in Section 3.3.1, suppose $V_S = \{v_1, \ldots, v_d\}$, let Φ be the $n \times d$ matrix $(v_1 \ \ldots \ v_d)$, and let $\phi(i)'$ be its ith row, i.e.,

$$\phi(i) = \big(v_{1,i}, \ldots, v_{d,i}\big)'.$$

In what follows, we consider iteration t of the algorithms. Let the state trajectory be $\mathcal{I}_t = (i_0, \ldots, i_\ell)$. Define affine functions $f_t, \psi_k \in \mathcal{A}_d[x]$, $k = 1, \ldots, \ell$, by

$$f_t(x) = \begin{pmatrix} f_{v_1,t}(x) \\ \vdots \\ f_{v_d,t}(x) \end{pmatrix} = \begin{pmatrix} \big\langle v_1 \,,\, \frac{G_t(x)}{t} \big\rangle \\ \vdots \\ \big\langle v_d \,,\, \frac{G_t(x)}{t} \big\rangle \end{pmatrix}, \qquad \psi_k(x) = \begin{pmatrix} \big\langle v_1 \,,\, Y_k(x) \big\rangle \\ \vdots \\ \big\langle v_d \,,\, Y_k(x) \big\rangle \end{pmatrix}. \tag{B.1}$$

As discussed in Section 3.3.1, Equation(t) of iteration t and Equation$(t;k)$ of sub-iteration k are stored after making the change of variable $x = \Phi r$ where $r \in \Re^d$. In particular, these equations are equivalent to:

$$\text{Equation}(t): \quad f_t(\Phi r) = 0, \qquad\qquad \text{Equation}(t;k): \quad \big(1 - \tfrac{1}{t}\big) f_{t-1}(\Phi r) + \tfrac{1}{t}\psi_k(\Phi r) = 0,$$

and the matrix representation of the functions in the left-hand sides of the equations are stored. Let the matrix representation of the function $f_t(\Phi r)$ be

$$f_t(\Phi r) = C_t r - b_t. \tag{B.2}$$

We derive the recursive formulas for computing the matrix C_t and vector b_t.

B.1 Derivations for Section 3.3.1

Based on steps 3 and 4 of Algorithm II, we have the relation

$$\begin{aligned} f_t(x) &= (1 - \tfrac{1}{t}) f_{t-1}(x) + \tfrac{1}{t}\psi_\ell(x) \\ &= (1 - \tfrac{1}{t}) f_{t-1}(x) + \tfrac{1}{t} \sum_{k=0}^{\ell-1} \big(\psi_{k+1}(x) - \psi_k(x)\big), \end{aligned}$$

where we define $\psi_0 \in \mathcal{A}_d[x]$ and $\psi_0(\cdot) \equiv 0$. Hence, if we have the following matrix representations of the functions,

$$\psi_{k+1}(\Phi r) - \psi_\ell(\Phi r) = U_k\, r - u_k, \qquad k = 0, \ldots, \ell - 1, \tag{B.3}$$

then C_t, b_t in the matrix representation (B.2) for $f_t(\Phi r)$ are given by

$$C_t = \big(1 - \tfrac{1}{t}\big) C_{t-1} + \tfrac{1}{t} \sum_{k=0}^{\ell-1} U_k, \qquad b_t = \big(1 - \tfrac{1}{t}\big) b_{t-1} + \tfrac{1}{t} \sum_{k=0}^{\ell-1} u_k. \tag{B.4}$$

We now calculate the matrices U_k and vectors u_k in Eq. (B.3). To this end, we calculate $\psi_{k+1}(x) - \psi_k(x)$. (We will make the change of variable $x = \Phi r$ at the end to obtain the desired expressions.)

For $k \le \ell$, we can express the function $Y_k(x)$ in familiar terms of temporal difference as: for every i,

$$Y_{k,i}(x) = \sum_{j=0}^{k-1} \delta\big\{i = i_j\big\} \cdot \sum_{m=j+1}^{k} \lambda^{m-1-j} \cdot \Big(\tilde{T}_{(\mathcal{I})_j^{m-1}} x - \tilde{T}_{(\mathcal{I})_j^m} x\Big), \tag{B.5}$$

where the mapping $\tilde{T}_{(\mathcal{I})_j^j}$ is defined as $\tilde{T}_{(\mathcal{I})_j^j} x = x_{i_j}$ for all x. By the definition of ψ_k [cf. Eq. (B.1)], we then have

$$\begin{aligned}\psi_k(x) &= \sum_{i=0}^{n} \phi(i) \cdot Y_{k,i}(x) = \sum_{i=0}^{n} \phi(i) \cdot \sum_{j=0}^{k-1} \delta\{i = i_j\} \cdot \sum_{m=j+1}^{k} \lambda^{m-1-j} \cdot \Big(\tilde{T}_{(\mathcal{I})_j^{m-1}} x - \tilde{T}_{(\mathcal{I})_j^m} x\Big) \\ &= \sum_{j=0}^{k-1} \phi(i_j) \cdot \sum_{m=j+1}^{k} \lambda^{m-1-j} \cdot \Big(\tilde{T}_{(\mathcal{I})_j^{m-1}} x - \tilde{T}_{(\mathcal{I})_j^m} x\Big).\end{aligned}$$

Hence, for $k \le \ell - 1$,

$$\begin{aligned}\psi_{k+1}(x) - \psi_k(x) &= \sum_{j=0}^{k} \lambda^{k-j} \cdot \phi(i_j) \cdot \Big(\tilde{T}_{(\mathcal{I})_j^k} x - \tilde{T}_{(\mathcal{I})_j^{k+1}} x\Big) \\ &= \left(\sum_{j=0}^{k} \lambda^{k-j} \cdot \phi(i_j) \cdot \beta^{k-j}\, \rho_{j,k}\right) \cdot \Big(x_{i_k} - \rho_{k,k+1}\, g_\mu(i_k, i_{k+1}) - \beta \rho_{k,k+1}\, x_{i_{k+1}}\Big),\end{aligned} \tag{B.6}$$

where, for ordinary PE [in which case $\tilde{T}_{\mathcal{I}}$ is defined by Eq. (3.1)],

$$\rho_{j,m} = 1, \qquad \forall\ j \le m \le \ell,$$

whereas for exploration-enhanced PE [in which case $\tilde{T}_{\mathcal{I}}$ is defined by Eq. (3.2)],

$$\rho_{j,j} = 1 \quad \text{and} \quad \rho_{j,m} = \textstyle\prod_{s=j}^{m-1} \frac{p_{i_s i_{s+1}}}{\bar{p}_{i_s i_{s+1}}}, \qquad \forall\ j < m \le \ell.$$

Let Z_{k+1} be the first term in the right-hand side of Eq. (B.6), i.e., let

$$Z_k = \sum_{j=0}^{k-1} \lambda^{k-1-j} \beta^{k-1-j} \rho_{j,k-1} \cdot \phi(i_j), \qquad k = 1, \ldots, \ell.$$

By this definition and Eq. (B.6), Z_k can be computed recursively and $\psi_{k+1} - \psi_k$ can be expressed in terms of Z_{k+1} as follows. With $Z_0 = 0$ and $\psi_0 \equiv 0$, for $k = 0, 1, \ldots, \ell - 1$,

$$Z_{k+1} = \lambda\beta\rho_{k-1,k} \cdot Z_k + \phi(i_k), \tag{B.7}$$

$$\psi_{k+1}(x) - \psi_k(x) = Z_{k+1} \cdot \big(x_{i_k} - \rho_{k,k+1}\, g_\mu(i_k, i_{k+1}) - \beta\rho_{k,k+1}\, x_{i_{k+1}}\big). \tag{B.8}$$

Finally, we make the change of variable $x = \Phi r$ in Eq. (B.8) to obtain

$$\psi_{k+1}(\Phi r) - \psi_k(\Phi r) = U_k r - u_k, \qquad k = 0, \ldots, \ell - 1,$$

where

$$U_k = Z_{k+1} \cdot \big(\phi(i_k) - \beta\rho_{k,k+1}\, \phi(i_{k+1})\big)', \qquad u_k = Z_{k+1} \cdot \rho_{k,k+1}\, g_\mu(i_k, i_{k+1}). \tag{B.9}$$

Equations (B.7), (B.9) together with (B.4) form recursive computation formulas for Algorithm II (as given in Section 3.3.1).

B.2 Derivations for Section 3.4

Variant A

To derive the recursive computation formulas of Variant A (Section 3.4), we reason as in Appendix B.1. Instead of Eq. (B.5), we now have

$$Y_{k,i}(x) = \sum_{j=0}^{k-1} \delta\{i = i_j\} \cdot \sum_{m=j+1}^{k} \lambda_i^{m-1-j} \cdot \Big(\tilde{T}_{(\mathcal{I})_j^{m-1}} x - \tilde{T}_{(\mathcal{I})_j^m} x\Big),$$

and

$$\psi_k(x) = \sum_{j=0}^{k-1} \phi(i_j) \cdot \sum_{m=j+1}^{k} \lambda_{i_j}^{m-1-j} \cdot \Big(\tilde{T}_{(\mathcal{I})_j^{m-1}} x - \tilde{T}_{(\mathcal{I})_j^m} x\Big).$$

So instead of Eq. (B.6), we now have

$$\psi_{k+1}(x) - \psi_k(x) = \left(\sum_{j=0}^{k} \lambda_{i_j}^{k-j} \cdot \phi(i_j) \cdot \beta^{k-j}\, \rho_{j,k}\right) \cdot \Big(x_{i_k} - \rho_{k,k+1}\, g_\mu(i_k, i_{k+1}) - \beta \rho_{k,k+1}\, x_{i_{k+1}}\Big). \quad \text{(B.10)}$$

As before, let Z_{k+1} be the first term in the right-hand side above; i.e., let

$$Z_k = \sum_{j=0}^{k-1} \lambda_{i_j}^{k-1-j} \beta^{k-1-j} \rho_{j,k-1} \cdot \phi(i_j), \qquad k = 1, \ldots, \ell.$$

Then Eq. (B.8) holds and the recursive formulas of Algorithm II hold for Variant A, except for the formula for Z_k, which we derive next.

Using the assumption that λ_i takes values in the set $\{\bar{\lambda}_1, \ldots, \bar{\lambda}_s\}$, we see that

$$\begin{aligned} Z_k &= \sum_{m=1}^{s} \sum_{j=0}^{k-1} \bar{\lambda}_m^{k-1-j} \cdot \delta\{\lambda_{i_j} = \bar{\lambda}_m\} \cdot \beta^{k-1-j} \rho_{j,k-1} \cdot \phi(i_j) \\ &= \sum_{m=1}^{s} Z_k^{(m)}, \end{aligned} \quad \text{(B.11)}$$

where for $m = 1, \ldots, s$, we define $Z_k^{(m)}$ by

$$Z_k^{(m)} = \sum_{j=0}^{k-1} \bar{\lambda}_m^{k-1-j} \cdot \beta^{k-1-j} \rho_{j,k-1} \cdot \delta\{\lambda_{i_j} = \bar{\lambda}_m\}\, \phi(i_j).$$

For each m, $Z_k^{(m)}$, $k \le \ell$, can be calculated recursively: with $Z_0^{(m)} = 0$,

$$Z_{k+1}^{(m)} = \bar{\lambda}_m \cdot \beta \rho_{k-1,k} \cdot Z_k^{(m)} + \delta\{\lambda_{i_k} = \bar{\lambda}_m\}\, \phi(i_k), \qquad k = 0, 1, \ldots, \ell - 1. \quad \text{(B.12)}$$

Equation (B.12) and (B.11), together with (B.9) and (B.4), form the recursive computation formulas for Variant A as given in Section 3.4.1.

Variant B

The formulas for Variant B can be similarly derived. Instead of Eq. (B.5), we now have

$$Y_{k,i}(x) = \sum_{j=0}^{k-1} \delta\big\{i = i_j\,,\, \lambda_i = \lambda_{i_0}\big\} \cdot \sum_{m=j+1}^{k} \lambda_i^{m-1-j} \cdot \Big(\tilde{T}_{(\mathcal{I})_j^{m-1}} x - \tilde{T}_{(\mathcal{I})_j^m} x\Big),$$

and

$$\psi_k(x) = \sum_{j=0}^{k-1} \phi(i_j)\,\delta\big\{\lambda_{i_j} = \lambda_{i_0}\big\} \cdot \sum_{m=j+1}^{k} \lambda_{i_0}^{m-1-j} \cdot \Big(\tilde{T}_{(\mathcal{I})_j^{m-1}} x - \tilde{T}_{(\mathcal{I})_j^m} x\Big).$$

So instead of Eq. (B.6), we have

$$\psi_{k+1}(x) - \psi_k(x) = \left(\sum_{j=0}^{k} \lambda_{i_0}^{k-j} \cdot \phi(i_j) \cdot \delta\big\{\lambda_{i_j} = \lambda_{i_0}\big\}\, \beta^{k-j}\, \rho_{j,k}\right) \cdot \Big(x_{i_k} - \rho_{k,k+1}\, g_\mu(i_k, i_{k+1}) - \beta \rho_{k,k+1}\, x_{i_{k+1}}\Big). \tag{B.13}$$

As before, we define Z_{k+1} to be the first term in the right-hand side above:

$$Z_k = \sum_{j=0}^{k-1} \lambda_{i_0}^{k-1-j} \beta^{k-1-j} \rho_{j,k-1} \cdot \delta\big\{\lambda_{i_j} = \lambda_{i_0}\big\}\, \phi(i_j), \qquad k = 1, \ldots, \ell.$$

They can be calculated recursively: with $Z_0 = 0$,

$$Z_{k+1} = \lambda_{i_0} \cdot \beta \rho_{k-1,k} \cdot Z_k + \delta\{\lambda_{i_k} = \lambda_{i_0}\}\, \phi(i_k), \qquad k = 1, \ldots, \ell - 1. \tag{B.14}$$

Equation (B.14), together with (B.9), (B.4), forms the recursive computation formulas for Variant B as given in Section 3.4.2.