HAL
archives-ouvertes.fr

Approximate modified policy iteration and its application to the game of Tetris

Bruno Scherrer, Mohammad Ghavamzadeh, Victor Gabillon, Boris Lesner, Matthieu Geist

▶ **To cite this version:**

Bruno Scherrer, Mohammad Ghavamzadeh, Victor Gabillon, Boris Lesner, Matthieu Geist. Approximate modified policy iteration and its application to the game of Tetris. Journal of Machine Learning Research, Journal of Machine Learning Research, 2015, 16, pp.1629–1676. <hal-01091341>

HAL Id: hal-01091341
https://hal.inria.fr/hal-01091341

Submitted on 8 Dec 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution| 4.0 International License

Journal of Machine Learning Research () Submitted 2013/07; Revised 2014/05; Published 2015/??

Approximate Modified Policy Iteration and its Application to the Game of Tetris

Bruno Scherrer[1] Bruno.Scherrer@inria.fr
Mohammad Ghavamzadeh[2] Mohammad.Ghavamzadeh@inria.fr
Victor Gabillon[3] Victor.Gabillon@inria.fr
Boris Lesner[1] Boris.Lesner@inria.fr
Matthieu Geist[4] Matthieu.Geist@supelec.fr
[1]*INRIA Nancy & Université de Lorraine - Team Maia, France*
[2]*Adobe Research, USA & INRIA Lille - Team SequeL, France*
[3]*INRIA Lille - Team SequeL, France*
[4]*Supélec - IMS-MaLIS Research Group, Metz, France*

Editor: Shie Mannor

Abstract

Modified policy iteration (MPI) is a dynamic programming (DP) algorithm that contains the two celebrated policy and value iteration methods. Despite its generality, MPI has not been thoroughly studied, especially its approximation form which is used when the state and/or action spaces are large or infinite. In this paper, we propose three implementations of approximate MPI (AMPI) that are extensions of the well-known approximate DP algorithms: fitted-value iteration, fitted-Q iteration, and classification-based policy iteration. We provide error propagation analysis that unify those for approximate policy and value iteration. We develop the finite-sample analysis of these algorithms, which highlights the influence of their parameters. In the classification-based version of the algorithm (CBMPI), the analysis shows that MPI's main parameter controls the balance between the estimation error of the classifier and the overall value function approximation. We illustrate and evaluate the behavior of these new algorithms in the Mountain Car and Tetris problems. Remarkably, in Tetris, CBMPI outperforms the existing DP approaches by a large margin, and competes with the current state-of-the-art methods while using fewer samples.[1]

Keywords: approximate dynamic programming, reinforcement learning, Markov decision processes, finite-sample analysis, performance bounds, game of tetris

1. Introduction

Modified Policy Iteration (MPI) (Puterman, 1994, Chapter 6, and references therein for a detailed historical account) is an iterative algorithm to compute the optimal policy and value function of a Markov Decision Process (MDP). Starting from an arbitrary value function

1. This paper is a significant extension of two conference papers by the authors (Scherrer *et al.*, 2012; Gabillon *et al.*, 2013). Here we discuss better the relation of the AMPI algorithms with other approximate DP methods, and provide more detailed description of the algorithms, proofs of the theorems, and report of the experimental results, especially in the game of Tetris. Moreover, we report new results in the game Tetris that were obtained after the publication of our paper on this topic (Gabillon *et al.*, 2013).

© Bruno Scherrer, Mohammad Ghavamzadeh, Victor Gabillon, Boris Lesner, Matthieu Geist.

v_0, it generates a sequence of value-policy pairs

$$\pi_{k+1} = \mathcal{G}\, v_k \qquad \text{(greedy step)} \qquad (1)$$
$$v_{k+1} = (T_{\pi_{k+1}})^m v_k \qquad \text{(evaluation step)} \qquad (2)$$

where $\mathcal{G}\, v_k$ is a *greedy* policy w.r.t. (with respect to) v_k, T_{π_k} is the Bellman operator associated to the policy π_k, and $m \geq 1$ is a parameter. MPI generalizes the well-known dynamic programming algorithms: Value Iteration (VI) and Policy Iteration (PI) for the values $m = 1$ and $m = \infty$, respectively. MPI has less computation per iteration than PI (in a way similar to VI), while enjoys the faster convergence (in terms of the number of iterations) of the PI algorithm (Puterman, 1994). In problems with large state and/or action spaces, approximate versions of VI (AVI) and PI (API) have been the focus of a rich literature (see *e.g.*, Bertsekas and Tsitsiklis 1996; Szepesvári 2010). Approximate VI (AVI) generates the next value function as the approximation of the application of the Bellman optimality operator to the current value (Singh and Yee, 1994; Gordon, 1995; Bertsekas and Tsitsiklis, 1996; Munos, 2007; Ernst *et al.*, 2005; Antos *et al.*, 2007; Munos and Szepesvári, 2008). On the other hand, approximate PI (API) first finds an approximation of the value of the current policy and then generates the next policy as greedy w.r.t. this approximation (Bertsekas and Tsitsiklis, 1996; Munos, 2003; Lagoudakis and Parr, 2003a; Lazaric *et al.*, 2010c, 2012). Another related algorithm is λ-policy iteration (Bertsekas and Ioffe, 1996), which is a rather complicated variation of MPI. It involves computing a fixed-point at each iteration, and thus, suffers from some of the drawbacks of the PI algorithms. This algorithm has been analyzed in its approximate form by Thiery and Scherrer (2010a); Scherrer (2013). The aim of this paper is to show that, similarly to its exact form, approximate MPI (AMPI) may represent an interesting alternative to AVI and API algorithms.

In this paper, we propose three implementations of AMPI (Section 3) that generalize the AVI implementations of Ernst *et al.* (2005); Antos *et al.* (2007); Munos and Szepesvári (2008) and the classification-based API algorithms of Lagoudakis and Parr (2003b); Fern *et al.* (2006); Lazaric *et al.* (2010a); Gabillon *et al.* (2011). We then provide an error propagation analysis of AMPI (Section 4), which shows how the L_p-norm of its performance loss

$$\ell_k = v_{\pi_*} - v_{\pi_k}$$

of using the policy π_k computed at some iteration k instead of the optimal policy π_* can be controlled through the errors at each iteration of the algorithm. We show that the error propagation analysis of AMPI is more involved than that of AVI and API. This is due to the fact that neither the contraction nor monotonicity arguments, that the error propagation analysis of these two algorithms rely on, hold for AMPI. The analysis of this section unifies those for AVI and API and is applied to the AMPI implementations presented in Section 3. We then detail the analysis of the three algorithms of Section 3 by providing their finite-sample analysis in Section 5. Interestingly, for the classification-based implementation of MPI (CBMPI), our analysis indicates that the parameter m allows us to balance the estimation error of the classifier with the overall quality of the value approximation. Finally, we evaluate the proposed algorithms and compare them with several existing methods in the Mountain Car and Tetris problems in Section 6. The game of Tetris is particularly challenging as the DP methods that are only based on approximating the value function

have performed poorly in this domain. An important contribution of this work is to show that the classification-based AMPI algorithm (CBMPI) outperforms the existing DP approaches by a large margin, and competes with the current state-of-the-art methods while using fewer samples.

2. Background

We consider a discounted MDP $\langle \mathcal{S}, \mathcal{A}, P, r, \gamma \rangle$, where $\mathcal{S}$ is a state space, $\mathcal{A}$ is a finite action space, $P(ds'|s,a)$, for all state-action pairs (s,a), is a probability kernel on $\mathcal{S}$, the reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is bounded by $R_{\max}$, and $\gamma \in (0,1)$ is a discount factor. A deterministic stationary policy (for short thereafter: a policy) is defined as a mapping $\pi : \mathcal{S} \to \mathcal{A}$. For a policy π, we may write $r_\pi(s) = r\big(s, \pi(s)\big)$ and $P_\pi(ds'|s) = P\big(ds'|s, \pi(s)\big)$. The value of the policy π in a state s is defined as the expected discounted sum of rewards received by starting at state s and then following the policy π, i.e.,

$$v_\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_\pi(s_t) \mid s_0 = s, s_{t+1} \sim P_\pi(\cdot|s_t)\right].$$

Similarly, the action-value function of a policy π at a state-action pair (s,a), $Q_\pi(s,a)$, is the expected discounted sum of rewards received by starting at state s, taking action a, and then following the policy π, i.e.,

$$Q_\pi(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a, s_{t+1} \sim P(\cdot|s_t, a_t), a_{t+1} = \pi(s_{t+1})\right].$$

Since the rewards are bounded by $R_{\max}$, the values and action-values are bounded by $V_{\max} = Q_{\max} = R_{\max}/(1-\gamma)$.

For any distribution μ on $\mathcal{S}$, μP_π is a distribution given by $(\mu P_\pi)(ds') = \int P_\pi(ds'|ds)\mu(ds)$. For any integrable function v on $\mathcal{S}$, $P_\pi v$ is a function defined as $(P_\pi v)(s) = \int v(s') P_\pi(ds'|s)$. The product of two kernels is naturally defined as $(P_{\pi'} P_\pi)(ds''|s) = \int P_{\pi'}(ds''|s') P_\pi(ds'|s)$. In analogy with the discrete space case, we write $(I - \gamma P_\pi)^{-1}$ to denote the kernel that is defined as $\sum_{t=0}^{\infty} (\gamma P_\pi)^t$.

The Bellman operator T_π of policy π takes an integrable function f on $\mathcal{S}$ as input and returns the function $T_\pi f$ defined as

$$\forall s \in \mathcal{S}, \quad [T_\pi f](s) = \mathbb{E}\big[r_\pi(s) + \gamma f(s') \mid s' \sim P_\pi(.|s)\big],$$

or in compact form, $T_\pi f = r_\pi + \gamma P_\pi f$. It is known that $v_\pi = (I - \gamma P_\pi)^{-1} r_\pi$ is the unique fixed-point of T_π. Given an integrable function f on $\mathcal{S}$, we say that a policy π is greedy w.r.t. f, and write $\pi = \mathcal{G}\, f$, if

$$\forall s \in \mathcal{S}, \quad [T_\pi f](s) = \max_a [T_a f](s),$$

or equivalently $T_\pi f = \max_{\pi'} [T_{\pi'} f]$. We denote by v_* the optimal value function. It is also known that v_* is the unique fixed-point of the Bellman optimality operator $T : v \to \max_\pi T_\pi v = T_{\mathcal{G}(v)} v$, and that a policy π_* that is greedy w.r.t. v_* is optimal and its value satisfies $v_{\pi_*} = v_*$.

We now define the concentrability coefficients (Munos, 2003, 2007; Munos and Szepesvári, 2008; Farahmand *et al.*, 2010; Scherrer, 2013) that measure the stochasticity of an MDP, and will later appear in our analysis. For any integrable function $f : \mathcal{S} \to \mathbb{R}$ and any distribution μ on $\mathcal{S}$, the μ-weighted L_p norm of f is defined as

$$\|f\|_{p,\mu} \triangleq \left[\int |f(x)|^p \mu(dx)\right]^{1/p}.$$

Given some distributions μ and ρ that will be clear in the context of the paper, for all integers i and q, we shall consider the following Radon-Nikodym derivative based quantities

$$c_q(j) \triangleq \max_{\pi_1,\ldots,\pi_j} \left\| \frac{d(\rho P_{\pi_1} P_{\pi_2} \cdots P_{\pi_j})}{d\mu} \right\|_{q,\mu}, \tag{3}$$

where $\pi_1, \ldots, \pi_j$ is any set of policies defined in the MDP, and with the understanding that if $\rho P_{\pi_1} P_{\pi_2} \cdots P_{\pi_j}$ is not absolutely continuous with respect to μ, then we take $c_q(j) = \infty$. These coefficients measure the mismatch between some reference measure μ and the distribution $\rho P_{\pi_1} P_{\pi_2} \cdots P_{\pi_j}$ obtained by starting the process from distribution ρ and then making j steps according to π_1, π_2, ... π_j, respectively. Since the bounds we shall derive will be based on these coefficients, they will be informative only if these coefficients are finite. We refer the reader to Munos (2007); Munos and Szepesvári (2008); Farahmand *et al.* (2010) for more discussion on this topic. In particular, the interested reader may find a simple MDP example for which these coefficients are reasonably small in Munos (2007, Section 5.5 and 7).

3. Approximate MPI Algorithms

In this section, we describe three approximate MPI (AMPI) algorithms. These algorithms rely on a function space $\mathcal{F}$ to approximate value functions, and in the third algorithm, also on a policy space Π to represent greedy policies. In what follows, we describe the iteration k of these iterative algorithms.

3.1 AMPI-V

The first and most natural AMPI algorithm presented in the paper, called AMPI-V, is described in Figure 1. In AMPI-V, we assume that the values v_k are represented in a function space $\mathcal{F} \subseteq \mathbb{R}^{\mathcal{S}}$. In any state s, the action $\pi_{k+1}(s)$ that is greedy w.r.t. v_k can be estimated as follows:

$$\pi_{k+1}(s) \in \arg\max_{a \in \mathcal{A}} \frac{1}{M} \sum_{j=1}^{M} \big(\widehat{T}_a^{(j)} v_k\big)(s), \tag{4}$$

$$\text{with} \qquad \big(\widehat{T}_a^{(j)} v_k\big)(s) = r_a^{(j)} + \gamma v_k(s_a^{(j)}),$$

where for all $a \in \mathcal{A}$ and $1 \le j \le M$, $r_a^{(j)}$ and $s_a^{(j)}$ are samples of rewards and next states when action a is taken in state s. Thus, approximating the greedy action in a state s requires $M|\mathcal{A}|$ samples. The algorithm works as follows. We sample N states from a distribution

Input: Value function space $\mathcal{F}$, state distribution μ
Initialize: Let $v_0 \in \mathcal{F}$ be an arbitrary value function
for $k = 0, 1, \ldots$ **do**
- ***Perform rollouts:***

Construct the rollout set $\mathcal{D}_k = \{s^{(i)}\}_{i=1}^{N},\ s^{(i)} \overset{\text{iid}}{\sim} \mu$
for all states $s^{(i)} \in \mathcal{D}_k$ **do**
Perform a rollout (using Equation 4 for each action)
$\widehat{v}_{k+1}(s^{(i)}) = \sum_{t=0}^{m-1} \gamma^t r_t^{(i)} + \gamma^m v_k(s_m^{(i)})$
end for
- ***Approximate value function:***

$v_{k+1} \in \operatorname{argmin}_{v \in \mathcal{F}} \widehat{\mathcal{L}}_k^{\mathcal{F}}(\widehat{\mu}; v)$ **(regression)** (see Equation 6)
end for

Figure 1: The pseudo-code of the AMPI-V algorithm.

μ on $\mathcal{S}$, and build a rollout set $\mathcal{D}_k = \{s^{(i)}\}_{i=1}^{N},\ s^{(i)} \sim \mu$. We denote by $\widehat{\mu}$ the empirical distribution corresponding to μ. From each state $s^{(i)} \in \mathcal{D}_k$, we generate a rollout of size m, *i.e.*, $\big(s^{(i)}, a_0^{(i)}, r_0^{(i)}, s_1^{(i)}, \ldots, a_{m-1}^{(i)}, r_{m-1}^{(i)}, s_m^{(i)}\big)$, where $a_t^{(i)}$ is the action suggested by π_{k+1} in state $s_t^{(i)}$, computed using Equation 4, and $r_t^{(i)}$ and $s_{t+1}^{(i)}$ are sampled reward and next state induced by this choice of action. For each $s^{(i)}$, we then compute a rollout estimate

$$\widehat{v}_{k+1}(s^{(i)}) = \sum_{t=0}^{m-1} \gamma^t r_t^{(i)} + \gamma^m v_k(s_m^{(i)}), \tag{5}$$

which is an unbiased estimate of $\big[(T_{\pi_{k+1}})^m v_k\big](s^{(i)})$. Finally, v_{k+1} is computed as the best fit in $\mathcal{F}$ to these estimates, *i.e.*, it is a function $v \in \mathcal{F}$ that minimizes the empirical error

$$\widehat{\mathcal{L}}_k^{\mathcal{F}}(\widehat{\mu}; v) = \frac{1}{N} \sum_{i=1}^{N} \big(\widehat{v}_{k+1}(s^{(i)}) - v(s^{(i)})\big)^2, \tag{6}$$

with the goal of minimizing the true error

$$\mathcal{L}_k^{\mathcal{F}}(\mu; v) = \Big|\Big|\big[(T_{\pi_{k+1}})^m v_k\big] - v\Big|\Big|_{2,\mu}^2 = \int \Big(\big[(T_{\pi_{k+1}})^m v_k\big](s) - v(s)\Big)^2 \mu(ds).$$

Each iteration of AMPI-V requires N rollouts of size m, and in each rollout, each of the $|\mathcal{A}|$ actions needs M samples to compute Equation 4. This gives a total of $Nm(M|\mathcal{A}| + 1)$ transition samples. Note that the fitted value iteration algorithm (Munos and Szepesvári, 2008) is a special case of AMPI-V when $m = 1$.

3.2 AMPI-Q

In AMPI-Q, we replace the value function $v : \mathcal{S} \to \mathbb{R}$ with the action-value function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Figure 2 contains the pseudocode of this algorithm. The Bellman operator for a policy π at a state-action pair (s, a) can then be written as

$$[T_\pi Q](s, a) = \mathbb{E}\big[r(s, a) + \gamma Q(s', \pi(s')) \mid s' \sim P(\cdot|s, a)\big],$$

and the greedy operator is defined as

$$\pi \in \mathcal{G}\, Q \iff \forall s, \quad \pi(s) = \arg\max_{a \in \mathcal{A}} Q(s,a).$$

In AMPI-Q, action-value functions Q_k are represented in a function space $\mathcal{F} \subseteq \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$, and the greedy action w.r.t. Q_k at a state s, *i.e.*, $\pi_{k+1}(s)$, is computed as

$$\pi_{k+1}(s) \in \arg\max_{a \in \mathcal{A}} Q_k(s,a). \tag{7}$$

The *evaluation step* is similar to that of AMPI-V, with the difference that now we work with state-action pairs. We sample N state-action pairs from a distribution μ on $\mathcal{S}\times\mathcal{A}$ and build a rollout set $\mathcal{D}_k = \{(s^{(i)}, a^{(i)})\}_{i=1}^N$, $(s^{(i)}, a^{(i)}) \sim \mu$. We denote by $\widehat{\mu}$ the empirical distribution corresponding to μ. For each $(s^{(i)}, a^{(i)}) \in \mathcal{D}_k$, we generate a rollout of size m, *i.e.*, $\big(s^{(i)}, a^{(i)}, r_0^{(i)}, s_1^{(i)}, a_1^{(i)}, \cdots, s_m^{(i)}, a_m^{(i)}\big)$, where the first action is $a^{(i)}$, $a_t^{(i)}$ for $t \geq 1$ is the action suggested by π_{k+1} in state $s_t^{(i)}$ computed using Equation 7, and $r_t^{(i)}$ and $s_{t+1}^{(i)}$ are sampled reward and next state induced by this choice of action. For each $(s^{(i)}, a^{(i)}) \in \mathcal{D}_k$, we then compute the rollout estimate

$$\widehat{Q}_{k+1}(s^{(i)}, a^{(i)}) = \sum_{t=0}^{m-1} \gamma^t r_t^{(i)} + \gamma^m Q_k(s_m^{(i)}, a_m^{(i)}),$$

which is an unbiased estimate of $\big[(T_{\pi_{k+1}})^m Q_k\big](s^{(i)}, a^{(i)})$. Finally, Q_{k+1} is the best fit to these estimates in $\mathcal{F}$, *i.e.*, it is a function $Q \in \mathcal{F}$ that minimizes the empirical error

$$\widehat{\mathcal{L}}_k^{\mathcal{F}}(\widehat{\mu}; Q) = \frac{1}{N}\sum_{i=1}^{N} \big(\widehat{Q}_{k+1}(s^{(i)}, a^{(i)}) - Q(s^{(i)}, a^{(i)})\big)^2, \tag{8}$$

with the goal of minimizing the true error

$$\mathcal{L}_k^{\mathcal{F}}(\mu; Q) = \Big|\Big|\big[(T_{\pi_{k+1}})^m Q_k\big] - Q\Big|\Big|_{2,\mu}^2 = \int \Big(\big[(T_{\pi_{k+1}})^m Q_k\big](s,a) - Q(s,a)\Big)^2 \mu(dsda).$$

Each iteration of AMPI-Q requires Nm samples, which is less than that for AMPI-V. However, it uses a hypothesis space on state-action pairs instead of states (a larger space than that used by AMPI-V). Note that the fitted-Q iteration algorithm (Ernst *et al.*, 2005; Antos *et al.*, 2007) is a special case of AMPI-Q when $m = 1$.

3.3 Classification-Based MPI

The third AMPI algorithm presented in this paper, called classification-based MPI (CBMPI), uses an explicit representation for the policies π_k, in addition to the one used for the value functions v_k. The idea is similar to the classification-based PI algorithms (Lagoudakis and Parr, 2003b; Fern *et al.*, 2006; Lazaric *et al.*, 2010a; Gabillon *et al.*, 2011) in which we search for the greedy policy in a policy space Π (defined by a classifier) instead of computing it from the estimated value or action-value function (similar to AMPI-V and AMPI-Q).
In order to describe CBMPI, we first rewrite the MPI formulation (Equations 1 and 2) as

$$v_k = (T_{\pi_k})^m v_{k-1} \qquad \text{(evaluation step)} \tag{9}$$

Input: Value function space $\mathcal{F}$, state distribution μ
Initialize: Let $Q_0 \in \mathcal{F}$ be an arbitrary value function
for $k = 0, 1, \ldots$ **do**
- ***Perform rollouts:***

Construct the rollout set $\mathcal{D}_k = \{(s^{(i)}, a^{(i)}\}_{i=1}^{N},\ (s^{(i)}, a^{(i)}) \overset{\text{iid}}{\sim} \mu$
for all states $(s^{(i)}, a^{(i)}) \in \mathcal{D}_k$ **do**
Perform a rollout (using Equation 7 for each action)
$\widehat{Q}_{k+1}(s^{(i)}, a^{(i)}) = \sum_{t=0}^{m-1} \gamma^t r_t^{(i)} + \gamma^m Q_k(s_m^{(i)}, a_m^{(i)})$,
end for
- ***Approximate action-value function:***

$Q_{k+1} \in \underset{Q \in \mathcal{F}}{\operatorname{argmin}}\, \widehat{\mathcal{L}}_k^{\mathcal{F}}(\widehat{\mu}; Q)$ **(regression)** (see Equation 8)
end for

Figure 2: The pseudo-code of the AMPI-Q algorithm.

Input: Value function space $\mathcal{F}$, policy space Π, state distribution μ
Initialize: Let $\pi_1 \in \Pi$ be an arbitrary policy and $v_0 \in \mathcal{F}$ an arbitrary value function
for $k = 1, 2, \ldots$ **do**
- ***Perform rollouts:***

Construct the rollout set $\mathcal{D}_k = \{s^{(i)}\}_{i=1}^{N},\ s^{(i)} \overset{\text{iid}}{\sim} \mu$
for all states $s^{(i)} \in \mathcal{D}_k$ **do**
Perform a rollout and return $\widehat{v}_k(s^{(i)})$ (using Equation 11)
end for
Construct the rollout set $\mathcal{D}'_k = \{s^{(i)}\}_{i=1}^{N'},\ s^{(i)} \overset{\text{iid}}{\sim} \mu$
for all states $s^{(i)} \in \mathcal{D}'_k$ and actions $a \in \mathcal{A}$ **do**
for $j = 1$ to M **do**
Perform a rollout and return $R_k^j(s^{(i)}, a)$ (using Equation 16)
end for
$\widehat{Q}_k(s^{(i)}, a) = \frac{1}{M} \sum_{j=1}^{M} R_k^j(s^{(i)}, a)$
end for
- ***Approximate value function:***

$v_k \in \underset{v \in \mathcal{F}}{\operatorname{argmin}}\, \widehat{\mathcal{L}}_k^{\mathcal{F}}(\widehat{\mu}; v)$ **(regression)** (see Equation 12)
- ***Approximate greedy policy:***

$\pi_{k+1} \in \underset{\pi \in \Pi}{\operatorname{argmin}}\, \widehat{\mathcal{L}}_k^{\Pi}(\widehat{\mu}; \pi)$ **(classification)** (see Equation 17)
end for

Figure 3: The pseudo-code of the CBMPI algorithm.

$$\pi_{k+1} = \mathcal{G}\left[(T_{\pi_k})^m v_{k-1}\right] \qquad \text{(greedy step)} \qquad (10)$$

Note that in this equivalent formulation both v_k and π_{k+1} are functions of $(T_{\pi_k})^m v_{k-1}$. CBMPI is an approximate version of this new formulation. As described in Figure 3, CBMPI begins with arbitrary initial policy $\pi_1 \in \Pi$ and value function $v_0 \in \mathcal{F}$.[2] At each

2. Note that the function space $\mathcal{F}$ and policy space Π are automatically defined by the choice of the regressor and classifier, respectively.

iteration k, a new value function v_k is built as the best approximation of the m-step Bellman operator $(T_{\pi_k})^m v_{k-1}$ in $\mathcal{F}$ (*evaluation step*). This is done by solving a regression problem whose target function is $(T_{\pi_k})^m v_{k-1}$. To set up the regression problem, we build a rollout set $\mathcal{D}_k$ by sampling N states i.i.d. from a distribution μ.[3] We denote by $\widehat{\mu}$ the empirical distribution corresponding to μ. For each state $s^{(i)} \in \mathcal{D}_k$, we generate a rollout $\big(s^{(i)}, a_0^{(i)}, r_0^{(i)}, s_1^{(i)}, \ldots, a_{m-1}^{(i)}, r_{m-1}^{(i)}, s_m^{(i)}\big)$ of size m, where $a_t^{(i)} = \pi_k(s_t^{(i)})$, and $r_t^{(i)}$ and $s_{t+1}^{(i)}$ are sampled reward and next state induced by this choice of action. From this rollout, we compute an unbiased estimate $\widehat{v}_k(s^{(i)})$ of $\big[(T_{\pi_k})^m v_{k-1}\big](s^{(i)})$ as in Equation 5:

$$\widehat{v}_k(s^{(i)}) = \sum_{t=0}^{m-1} \gamma^t r_t^{(i)} + \gamma^m v_{k-1}(s_m^{(i)}), \tag{11}$$

and use it to build a training set $\big\{\big(s^{(i)}, \widehat{v}_k(s^{(i)})\big)\big\}_{i=1}^N$. This training set is then used by the regressor to compute v_k as an estimate of $(T_{\pi_k})^m v_{k-1}$. Similar to the AMPI-V algorithm, the regressor here finds a function $v \in \mathcal{F}$ that minimizes the empirical error

$$\widehat{\mathcal{L}}_k^{\mathcal{F}}(\widehat{\mu}; v) = \frac{1}{N} \sum_{i=1}^N \big(\widehat{v}_k(s^{(i)}) - v(s^{(i)})\big)^2, \tag{12}$$

with the goal of minimizing the true error

$$\mathcal{L}_k^{\mathcal{F}}(\mu; v) = \Big|\Big|\big[(T_{\pi_k})^m v_{k-1}\big] - v\Big|\Big|_{2,\mu}^2 = \int \Big(\big[(T_{\pi_k})^m v_{k-1}\big](s) - v(s)\Big)^2 \mu(ds).$$

The *greedy step* at iteration k computes the policy π_{k+1} as the best approximation of $\mathcal{G}\big[(T_{\pi_k})^m v_{k-1}\big]$ by solving a cost-sensitive classification problem. From the definition of a greedy policy, if $\pi = \mathcal{G}\big[(T_{\pi_k})^m v_{k-1}\big]$, for each $s \in \mathcal{S}$, we have

$$\big[T_\pi (T_{\pi_k})^m v_{k-1}\big](s) = \max_{a \in \mathcal{A}} \big[T_a (T_{\pi_k})^m v_{k-1}\big](s). \tag{13}$$

By defining $Q_k(s,a) = \big[T_a (T_{\pi_k})^m v_{k-1}\big](s)$, we may rewrite Equation 13 as

$$Q_k\big(s, \pi(s)\big) = \max_{a \in \mathcal{A}} Q_k(s,a). \tag{14}$$

The cost-sensitive error function used by CBMPI is of the form

$$\mathcal{L}^{\Pi}_{\pi_k, v_{k-1}}(\mu; \pi) = \int \Big[\max_{a \in \mathcal{A}} Q_k(s,a) - Q_k\big(s, \pi(s)\big)\Big] \mu(ds). \tag{15}$$

To simplify the notation we use $\mathcal{L}_k^{\Pi}$ instead of $\mathcal{L}^{\Pi}_{\pi_k, v_{k-1}}$. To set up this cost-sensitive classification problem, we build a rollout set $\mathcal{D}'_k$ by sampling N' states i.i.d. from a distribution μ. For each state $s^{(i)} \in \mathcal{D}'_k$ and each action $a \in \mathcal{A}$, we build M independent rollouts of size $m+1$, *i.e.*,[4]

$$\Big(s^{(i)}, a, r_0^{(i,j)}, s_1^{(i,j)}, a_1^{(i,j)}, \ldots, a_m^{(i,j)}, r_m^{(i,j)}, s_{m+1}^{(i,j)}\Big)_{j=1}^M,$$

3. Here we used the same sampling distribution μ for both regressor and classifier, but in general different distributions may be used for these two components of the algorithm.
4. In practice, one may implement CBMPI in more sample-efficient way by reusing the rollouts generated for the greedy step in the evaluation step, but we do not consider this here because it makes the forthcoming analysis more complicated.

where for $t \geq 1$, $a_t^{(i,j)} = \pi_k(s_t^{(i,j)})$, and $r_t^{(i,j)}$ and $s_{t+1}^{(i,j)}$ are sampled reward and next state induced by this choice of action. From these rollouts, we compute an unbiased estimate of $Q_k(s^{(i)}, a)$ as $\widehat{Q}_k(s^{(i)}, a) = \frac{1}{M}\sum_{j=1}^{M} R_k^j(s^{(i)}, a)$ where

$$R_k^j(s^{(i)}, a) = \sum_{t=0}^{m} \gamma^t r_t^{(i,j)} + \gamma^{m+1} v_{k-1}(s_{m+1}^{(i,j)}). \tag{16}$$

Given the outcome of the rollouts, CBMPI uses a cost-sensitive classifier to return a policy π_{k+1} that minimizes the following *empirical error*

$$\widehat{\mathcal{L}}_k^{\Pi}(\widehat{\mu}; \pi) = \frac{1}{N'} \sum_{i=1}^{N'} \Big[\max_{a \in \mathcal{A}} \widehat{Q}_k(s^{(i)}, a) - \widehat{Q}_k\big(s^{(i)}, \pi(s^{(i)})\big) \Big], \tag{17}$$

with the goal of minimizing the true error $\mathcal{L}_k^{\Pi}(\mu; \pi)$ defined by Equation 15.

Each iteration of CBMPI requires $Nm + M|\mathcal{A}|N'(m+1)$ (or $M|\mathcal{A}|N'(m+1)$ in case we reuse the rollouts, see Footnote 4) transition samples. Note that when m tends to ∞, we recover the DPI algorithm proposed and analyzed by Lazaric *et al.* (2010a).

3.4 Possible Approaches to Reuse the Samples

In all the proposed AMPI algorithms, we generate fresh samples for the rollouts, and even for the starting states, at each iteration. This may result in relatively high sample complexity for these algorithms. In this section, we propose two possible approaches to circumvent this problem and to keep the number of samples independent of the number of iterations.

One approach would be to use a fixed set of starting samples $(s^{(i)})$ or $(s^{(i)}, a^{(i)})$ for all iterations, and think of a tree of depth m that contains all the possible outcomes of m-steps choices of actions (this tree contains $|A|^m$ leaves). Using this tree, all the trajectories with the same actions share the same samples. In practice, it is not necessarily to build the entire depth m tree, it is only needed to add a branch when the desired action does not belong to the tree. Using this approach, that is reminiscent of the work by Kearns *et al.* (2000), the sample complexity of the algorithm no longer depends on the number of iterations. For example, we may only need $NM|A|^m$ transitions for the CBMPI algorithm.

We may also consider the case where we do not have access to a generative model of the system, and all we have is a set of trajectories of size m generated by a behavior policy π_b that is assumed to choose all actions a in each state s with a positive probability (*i.e.*, $\pi_b(a|s) > 0$, $\forall s, \forall a$) (Precup *et al.*, 2000, 2001; Geist and Scherrer, 2014). In this case, one may still compute an unbiased estimate of the application of $(T_\pi)^m$ operator to value and action-value functions. For instance, given a m-step sample trajectory $(s, a_0, r_0, s_1, \ldots, s_m, a_m)$ generated by π_b, an unbiased estimate of $[(T_\pi)^m v](s)$ may be computed as (assuming that the distribution μ has the following factored form $p(s, a_0|\mu) = p(s)\pi_b(a_0|s)$ at state s)

$$y = \sum_{t=0}^{m-1} \alpha_t \gamma^t r_t + \alpha_m \gamma^m v(s_m), \qquad \text{where} \qquad \alpha_t = \prod_{j=1}^{t} \frac{1_{a_j = \pi(s_j)}}{\pi_b(a_j|s_j)}$$

is an importance sampling correction factor that can be computed along the trajectory. Note that this process may increase the variance of such an estimate, and thus, requires many more samples to be accurate – the price to pay for the absence of a generative model.

4. Error Propagation

In this section, we derive a general formulation for propagation of errors through the iterations of an AMPI algorithm. The line of analysis for error propagation is different in VI and PI algorithms. VI analysis is based on the fact that this algorithm computes the fixed point of the Bellman optimality operator, and this operator is a γ-contraction in max-norm (Bertsekas and Tsitsiklis, 1996; Munos, 2007). On the other hand, it can be shown that the operator by which PI updates the value from one iteration to the next is not a contraction in max-norm in general. Unfortunately, we can show that the same property holds for MPI when it does not reduce to VI (*i.e.*, for $m > 1$).

Proposition 1 *If $m > 1$, there exists no norm for which the operator that MPI uses to update the values from one iteration to the next is a contraction.*

Proof We consider the MDP with two states $\{s_1, s_2\}$, two actions $\{change, stay\}$, rewards $r(s_1) = 0,\ r(s_2) = 1$, and transitions $P_{ch}(s_2|s_1) = P_{ch}(s_1|s_2) = P_{st}(s_1|s_1) = P_{st}(s_2|s_2) = 1$. Consider two value functions $v = (\epsilon, 0)$ and $v' = (0, \epsilon)$ with $\epsilon > 0$. Their corresponding greedy policies are $\pi = (st, ch)$ and $\pi' = (ch, st)$, and the next iterates of v and v' can be computed as $(T_\pi)^m v = \begin{pmatrix} \gamma^m \epsilon \\ 1 + \gamma^m \epsilon \end{pmatrix}$ and $(T_{\pi'})^m v' = \begin{pmatrix} \frac{\gamma - \gamma^m}{1-\gamma} + \gamma^m \epsilon \\ \frac{1-\gamma^m}{1-\gamma} + \gamma^m \epsilon \end{pmatrix}$. Thus, $(T_{\pi'})^m v' - (T_\pi)^m v = \begin{pmatrix} \frac{\gamma - \gamma^m}{1-\gamma} \\ \frac{\gamma - \gamma^m}{1-\gamma} \end{pmatrix}$, while $v' - v = \begin{pmatrix} -\epsilon \\ \epsilon \end{pmatrix}$. Since ϵ can be arbitrarily small, the norm of $(T_{\pi'})^m v' - (T_\pi)^m v$ can be arbitrarily larger than the norm of $v - v'$ as long as $m > 1$. ■

We also know that the analysis of PI usually relies on the fact that the sequence of the generated values is non-decreasing (Bertsekas and Tsitsiklis, 1996; Munos, 2003). Unfortunately, it can be easily shown that for m finite, the value functions generated by MPI may decrease (it suffices to take a very high initial value). It can be seen from what we just described and Proposition 1 that for $m \neq 1$ and ∞, MPI is neither contracting nor non-decreasing, and thus, a new proof is needed for the propagation of errors in this algorithm.

To study error propagation in AMPI, we introduce an abstract algorithmic model that accounts for potential errors. AMPI starts with an arbitrary value v_0 and at each iteration $k \geq 1$ computes the greedy policy w.r.t. v_{k-1} with some error ϵ'_k, called the *greedy step error*. Thus, we write the new policy π_k as

$$\pi_k = \widehat{\mathcal{G}}_{\epsilon'_k} v_{k-1}. \tag{18}$$

Equation 18 means that for any policy π', we have $T_{\pi'} v_{k-1} \leq T_{\pi_k} v_{k-1} + \epsilon'_k$. AMPI then generates the new value function v_k with some error ϵ_k, called the *evaluation step error*

$$v_k = (T_{\pi_k})^m v_{k-1} + \epsilon_k. \tag{19}$$

Before showing how these two errors are propagated through the iterations of AMPI, let us first define them in the context of each of the algorithms presented in Section 3 separately.

AMPI-V: The term ϵ_k is the error when fitting the value function v_k. This error can be further decomposed into two parts: the one related to the approximation power of $\mathcal{F}$ and

the one due to the finite number of samples/rollouts. The term ϵ'_k is the error due to using a finite number of samples M for estimating the greedy actions.

AMPI-Q: In this case $\epsilon'_k = 0$ and ϵ_k is the error in fitting the state-action value function Q_k.

CBMPI: This algorithm iterates as follows:

$$v_k = (T_{\pi_k})^m v_{k-1} + \epsilon_k$$
$$\pi_{k+1} = \widehat{\mathcal{G}}_{\epsilon'_{k+1}} \left[(T_{\pi_k})^m v_{k-1} \right].$$

Unfortunately, this does not exactly match the model described in Equations 18 and 19. By introducing the auxiliary variable $w_k \stackrel{\Delta}{=} (T_{\pi_k})^m v_{k-1}$, we have $v_k = w_k + \epsilon_k$, and thus, we may write

$$\pi_{k+1} = \widehat{\mathcal{G}}_{\epsilon'_{k+1}} \left[w_k \right]. \tag{20}$$

Using $v_{k-1} = w_{k-1} + \epsilon_{k-1}$, we have

$$w_k = (T_{\pi_k})^m v_{k-1} = (T_{\pi_k})^m (w_{k-1} + \epsilon_{k-1}) = (T_{\pi_k})^m w_{k-1} + (\gamma P_{\pi_k})^m \epsilon_{k-1}. \tag{21}$$

Now, Equations 20 and 21 exactly match Equations 18 and 19 by replacing v_k with w_k and ϵ_k with $(\gamma P_{\pi_k})^m \epsilon_{k-1}$.

The rest of this section is devoted to show how the errors ϵ_k and ϵ'_k propagate through the iterations of an AMPI algorithm. We only outline the main arguments that will lead to the performance bounds of Theorems 7 and 8 and report most technical details of the proof in Appendices A to C. To do this, we follow the line of analysis developed by Scherrer and Thiéry (2010), and consider the following three quantities:

1) The distance between the optimal value function and the value before approximation at the k^{th} iteration:

$$d_k \stackrel{\Delta}{=} v_* - (T_{\pi_k})^m v_{k-1} = v_* - (v_k - \epsilon_k).$$

2) The shift between the value before approximation and the value of the policy at the k^{th} iteration:

$$s_k \stackrel{\Delta}{=} (T_{\pi_k})^m v_{k-1} - v_{\pi_k} = (v_k - \epsilon_k) - v_{\pi_k}.$$

3) The (approximate) Bellman residual at the k^{th} iteration:

$$b_k \stackrel{\Delta}{=} v_k - T_{\pi_{k+1}} v_k.$$

We are interested in finding an upper bound on the **loss**

$$l_k \stackrel{\Delta}{=} v_* - v_{\pi_k} = d_k + s_k.$$

To do so, we will upper bound d_k and s_k, which requires a bound on the Bellman residual b_k. More precisely, the core of our analysis is to prove the following point-wise inequalities for our three quantities of interest.

Lemma 2 *Let $k \geq 1$, $x_k \stackrel{\Delta}{=} (I - \gamma P_{\pi_k})\epsilon_k + \epsilon'_{k+1}$ and $y_k \stackrel{\Delta}{=} -\gamma P_{\pi_*}\epsilon_k + \epsilon'_{k+1}$. We have:*

$$\begin{aligned} b_k &\leq (\gamma P_{\pi_k})^m b_{k-1} + x_k, \\ d_{k+1} &\leq \gamma P_{\pi_*} d_k + y_k + \sum_{j=1}^{m-1} (\gamma P_{\pi_{k+1}})^j b_k, \\ s_k &= (\gamma P_{\pi_k})^m (I - \gamma P_{\pi_k})^{-1} b_{k-1}. \end{aligned}$$

Proof See Appendix A. ■

Since the stochastic kernels are non-negative, the bounds in Lemma 2 indicate that the loss l_k will be bounded if the errors ϵ_k and ϵ'_k are controlled. In fact, if we define ϵ as a uniform upper-bound on the pointwise absolute value of the errors, $|\epsilon_k|$ and $|\epsilon'_k|$, the first inequality in Lemma 2 implies that $b_k \leq O(\epsilon)$, and as a result, the second and third inequalities gives us $d_k \leq O(\epsilon)$ and $s_k \leq O(\epsilon)$. This means that the loss will also satisfy $l_k \leq O(\epsilon)$.

Our bound for the loss l_k is the result of careful expansion and combination of the three inequalities in Lemma 2. Before we state this result, we introduce some notations that will ease our formulation and significantly simplify our proofs compared to those in the similar existing work (Munos, 2003, 2007; Scherrer, 2013).

Definition 3 *For a positive integer n, we define $\mathbb{P}_n$ as the smallest set of discounted transition kernels that are defined as follows:*

1) *for any set of n policies $\{\pi_1, \ldots, \pi_n\}$, $(\gamma P_{\pi_1})(\gamma P_{\pi_2}) \ldots (\gamma P_{\pi_n}) \in \mathbb{P}_n$,*

2) *for any $\alpha \in (0, 1)$ and $(P_1, P_2) \in \mathbb{P}_n \times \mathbb{P}_n$, $\alpha P_1 + (1 - \alpha) P_2 \in \mathbb{P}_n$.*

Furthermore, we use the somewhat abusive notation Γ^n for denoting any element of $\mathbb{P}_n$. For example, if we write a transition kernel P as $P = \alpha_1 \Gamma^i + \alpha_2 \Gamma^j \Gamma^k = \alpha_1 \Gamma^i + \alpha_2 \Gamma^{j+k}$, it should be read as: "there exist $P_1 \in \mathbb{P}_i$, $P_2 \in \mathbb{P}_j$, $P_3 \in \mathbb{P}_k$, and $P_4 \in \mathbb{P}_{k+j}$ such that $P = \alpha_1 P_1 + \alpha_2 P_2 P_3 = \alpha_1 P_1 + \alpha_2 P_4$."

Using the notation in Definition 3, we now derive a point-wise bound on the loss.

Lemma 4 *After k iterations, the losses of AMPI-V and AMPI-Q satisfy*

$$l_k \leq 2 \sum_{i=1}^{k-1} \sum_{j=i}^{\infty} \Gamma^j |\epsilon_{k-i}| + \sum_{i=0}^{k-1} \sum_{j=i}^{\infty} \Gamma^j |\epsilon'_{k-i}| + h(k),$$

while the loss of CBMPI satisfies

$$l_k \leq 2 \sum_{i=1}^{k-2} \sum_{j=i+m}^{\infty} \Gamma^j |\epsilon_{k-i-1}| + \sum_{i=0}^{k-1} \sum_{j=i}^{\infty} \Gamma^j |\epsilon'_{k-i}| + h(k),$$

where $h(k) \stackrel{\Delta}{=} 2 \sum_{j=k}^{\infty} \Gamma^j |d_0|$ or $h(k) \stackrel{\Delta}{=} 2 \sum_{j=k}^{\infty} \Gamma^j |b_0|$.

Proof See Appendix B. ■

Remark 5 *A close look at the existing point-wise error bounds for AVI (Munos, 2007, Lemma 4.1) and API (Munos, 2003, Corollary 10) shows that they do not consider error in the greedy step (*i.e., $\epsilon'_k = 0$*) and have the following form:*

$$\limsup_{k\to\infty} l_k \leq 2 \limsup_{k\to\infty} \sum_{i=1}^{k-1} \sum_{j=i}^{\infty} \Gamma^j |\epsilon_{k-i}|.$$

This indicates that the bound in Lemma 4 not only unifies the analysis of AVI and API, but it generalizes them to the case of error in the greedy step and to a finite number of iterations k. Moreover, our bound suggests that the way the errors are propagated in the whole family of algorithms, VI/PI/MPI, is independent of m at the level of abstraction suggested by Definition 3.[5]

An important immediate consequence of the point-wise bound of Lemma 4 is a simple guarantee on the performance of the algorithms. Let us define $\epsilon = \sup_{j\geq 1} \|\epsilon_j\|_\infty$ and $\epsilon' = \sup_{j\geq 1} \|\epsilon'_j\|_\infty$ as uniform bounds on the evaluation and greedy step errors. Now by taking the max-norm (using the fact that for all i, $\|\Gamma^i\|_\infty = \gamma^i$) and limsup when k tends to infinity, we obtain

$$\limsup_{k\to\infty} \|l_k\|_\infty \leq \frac{2\gamma\epsilon + \epsilon'}{(1-\gamma)^2}. \tag{22}$$

Such a bound is a generalization of the bounds for AVI ($m = 1$ and $\epsilon' = 0$) and API ($m = \infty$) in Bertsekas and Tsitsiklis (1996). This bound can be read as follows: if we can control the max-norm of the evaluation and greedy errors at all iterations, then we can control the loss of the policy returned by the algorithm w.r.t. the optimal policy. Conversely, another interpretation of the above bound is that errors should not be too big if we want to have a performance guarantee. Since the loss is alway bounded by $2V_{\max}$, the bound stops to be informative as soon as $2\gamma\epsilon + \epsilon' > 2(1-\gamma)^2 V_{\max} = 2(1-\gamma)R_{\max}$.

Assume we use (max-norm) regression and classification for the evaluation and greedy steps. Then, the above result means that one can make a *reduction* from the RL problem to these regression and classification problems. Furthermore, if any significant breakthrough is made in the literature for these (more standard problems), the RL setting can automatically benefit from it. The error terms ϵ and ϵ' in the above bound are expressed in terms of the max-norm. Since most regressors and classifiers, including those we have described in the algorithms, control some weighted quadratic norm, the practical range of a result like Equation 22 is limited. The rest of this section addresses this specific issue, by developing a somewhat more complicated but more useful error analysis in L_p-norm.

We now turn the point-wise bound of Lemma 4 into a bound in weighted L_p-norm, which we recall, for any function $f : \mathcal{S} \to \mathbb{R}$ and any distribution μ on $\mathcal{S}$ is defined as $\|f\|_{p,\mu} \stackrel{\Delta}{=} \left[\int |f(x)|^p \mu(dx)\right]^{1/p}$. Munos (2003, 2007); Munos and Szepesvári (2008), and the recent work of Farahmand *et al.* (2010), which provides the most refined bounds for API and AVI, show how to do this process through quantities, called *concentrability coefficients*. These coefficients use the Radon-Nikodym coefficients introduced in Section 2 and measure

5. Note however that the dependence on m will reappear if we make explicit what is hidden in Γ^j terms.

how a distribution over states may concentrate through the dynamics of the MDP. We now state a technical lemma that allows to convert componentwise bounds to L_p-norm bounds, and that generalizes the analysis of Farahmand *et al.* (2010) to a larger class of concentrability coefficients.

Lemma 6 *Let $\mathcal{I}$ and $(\mathcal{J}_i)_{i\in\mathcal{I}}$ be sets of positive integers, $\{\mathcal{I}_1,\dots,\mathcal{I}_n\}$ be a partition of $\mathcal{I}$, and f and $(g_i)_{i\in\mathcal{I}}$ be functions satisfying*

$$|f| \le \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}_i}\Gamma^j|g_i| = \sum_{l=1}^{n}\sum_{i\in\mathcal{I}_l}\sum_{j\in\mathcal{J}_i}\Gamma^j|g_i|.$$

Then for all p, q and q' such that $\frac{1}{q}+\frac{1}{q'}=1$, and for all distributions ρ and μ, we have

$$\|f\|_{p,\rho} \le \sum_{l=1}^{n}\left(\mathcal{C}_q(l)\right)^{1/p}\sup_{i\in\mathcal{I}_l}\|g_i\|_{pq',\mu}\sum_{i\in\mathcal{I}_l}\sum_{j\in\mathcal{J}_i}\gamma^j,$$

with the following concentrability coefficients

$$\mathcal{C}_q(l) \triangleq \frac{\sum_{i\in\mathcal{I}_l}\sum_{j\in\mathcal{J}_i}\gamma^j c_q(j)}{\sum_{i\in\mathcal{I}_l}\sum_{j\in\mathcal{J}_i}\gamma^j},$$

where $c_q(j)$ is defined by Equation 3.

Proof See Appendix C. ■

We now derive an L_p-norm bound for the loss of the AMPI algorithm by applying Lemma 6 to the point-wise bound of Lemma 4.

Theorem 7 *For all q, l, k and d, define the following concentrability coefficients:*

$$\mathcal{C}_q^{l,k,d} \triangleq \frac{(1-\gamma)^2}{\gamma^l-\gamma^k}\sum_{i=l}^{k-1}\sum_{j=i}^{\infty}\gamma^j c_q(j+d),$$

with $c_q(j)$ given by Equation 3. Let ρ and μ be distributions over states. Let p, q, and q' be such that $\frac{1}{q}+\frac{1}{q'}=1$. After k iterations, the loss of AMPI satisfies

$$\|l_k\|_{p,\rho} \le 2\sum_{i=1}^{k-1}\frac{\gamma^i}{1-\gamma}\left(\mathcal{C}_q^{i,i+1,0}\right)^{\frac{1}{p}}\|\epsilon_{k-i}\|_{pq',\mu} + \sum_{i=0}^{k-1}\frac{\gamma^i}{1-\gamma}\left(\mathcal{C}_q^{i,i+1,0}\right)^{\frac{1}{p}}\|\epsilon'_{k-i}\|_{pq',\mu} + g(k),$$

while the loss of CBMPI satisfies

$$\|l_k\|_{p,\rho} \le 2\gamma^m\sum_{i=1}^{k-2}\frac{\gamma^i}{1-\gamma}\left(\mathcal{C}_q^{i,i+1,m}\right)^{\frac{1}{p}}\|\epsilon_{k-i-1}\|_{pq',\mu} + \sum_{i=0}^{k-1}\frac{\gamma^i}{1-\gamma}\left(\mathcal{C}_q^{i,i+1,0}\right)^{\frac{1}{p}}\|\epsilon'_{k-i}\|_{pq',\mu} + g(k),$$

where $g(k) \triangleq \frac{2\gamma^k}{1-\gamma}\left(\mathcal{C}_q^{k,k+1,0}\right)^{\frac{1}{p}}\min\left(\|d_0\|_{pq',\mu},\|b_0\|_{pq',\mu}\right)$.

Proof We only detail the proof for AMPI, the proof is similar for CBMPI. We define $\mathcal{I} = \{1, 2, \ldots, 2k\}$ and the (trivial) partition $\mathcal{I} = \{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_{2k}\}$, where $\mathcal{I}_i = \{i\}$, $i \in \{1, \ldots, 2k\}$. For each $i \in \mathcal{I}$, we also define

$$g_i = \begin{cases} 2\epsilon_{k-i} & \text{if} \quad 1 \le i \le k-1, \\ \epsilon'_{k-(i-k)} & \text{if} \quad k \le i \le 2k-1, \\ 2d_0 \text{ (or } 2b_0) & \text{if} \quad i = 2k, \end{cases}$$

$$\text{and} \quad \mathcal{J}_i = \begin{cases} \{i, \cdots\} & \text{if} \quad 1 \le i \le k-1, \\ \{i-k \cdots\} & \text{if} \quad k \le i \le 2k-1, \\ \{k\} & \text{if} \quad i = 2k. \end{cases}$$

With the above definitions and the fact that the loss l_k is non-negative, Lemma 4 may be rewritten as

$$|l_k| \le \sum_{l=1}^{2k} \sum_{i \in \mathcal{I}_l} \sum_{j \in \mathcal{J}_i} \Gamma^j |g_i|.$$

The result follows by applying Lemma 6 and noticing that $\sum_{i=i_0}^{k-1} \sum_{j=i}^{\infty} \gamma^j = \frac{\gamma^{i_0} - \gamma^k}{(1-\gamma)^2}$. ■

Similar to the results of Farahmand *et al.* (2010), this bound shows that the last iterations have the highest influence on the loss and the influence decreases at the exponential rate γ towards the initial iterations. This phenomenon is related to the fact that the DP algorithms progressively forget about the past iterations. This is similar to the fact that exact VI and PI converge to the optimal limit independently of their initialization.

We can group the terms differently and derive an alternative L_p-norm bound for the loss of AMPI and CBMPI. This also shows the flexibility of Lemma 6 for turning the point-wise bound of Lemma 4 into L_p-norm bounds.

Theorem 8 *With the notations of Theorem 7, and writing* $\epsilon = \sup_{1 \le j \le k-1} \|\epsilon_j\|_{pq',\mu}$ *and* $\epsilon' = \sup_{1 \le j \le k} \|\epsilon'_j\|_{pq',\mu}$*, the loss of AMPI satisfies*

$$\|l_k\|_{p,\rho} \le \frac{2(\gamma - \gamma^k) \left(\mathcal{C}_q^{1,k,0}\right)^{\frac{1}{p}}}{(1-\gamma)^2} \epsilon + \frac{(1-\gamma^k) \left(\mathcal{C}_q^{0,k,0}\right)^{\frac{1}{p}}}{(1-\gamma)^2} \epsilon' + g(k), \tag{23}$$

while the loss of CBMPI satisfies

$$\|l_k\|_{p,\rho} \le \frac{2\gamma^m(\gamma - \gamma^{k-1}) \left(\mathcal{C}_q^{2,k,m}\right)^{\frac{1}{p}}}{(1-\gamma)^2} \epsilon + \frac{(1-\gamma^k) \left(\mathcal{C}_q^{0,k,0}\right)^{\frac{1}{p}}}{(1-\gamma)^2} \epsilon' + g(k). \tag{24}$$

Proof We only give the details of the proof for AMPI, the proof is similar for CBMPI. Defining $\mathcal{I} = \{1, 2, \cdots, 2k\}$ and g_i as in the proof of Theorem 7, we now consider the partition $\mathcal{I} = \{\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3\}$ as $\mathcal{I}_1 = \{1, \ldots, k-1\}$, $\mathcal{I}_2 = \{k, \ldots, 2k-1\}$, and $\mathcal{I}_3 = \{2k\}$, where for each $i \in \mathcal{I}$

$$\mathcal{J}_i = \begin{cases} \{i, i+1, \cdots\} & \text{if} \quad 1 \le i \le k-1, \\ \{i-k, i-k+1, \cdots\} & \text{if} \quad k \le i \le 2k-1, \\ \{k\} & \text{if} \quad i = 2k. \end{cases}$$

The proof ends similar to that of Theorem 7. ■

By sending the iteration number k to infinity, one obtains the following bound for AMPI:

$$\limsup_{k\to\infty} \|l_k\|_{p,\rho} \leq \frac{2\gamma \left(\mathcal{C}_q^{1,\infty,0}\right)^{\frac{1}{p}} \epsilon + \left(\mathcal{C}_q^{0,\infty,0}\right)^{\frac{1}{p}} \epsilon'}{(1-\gamma)^2}.$$

Compared to the simple max-norm bound of Equation 22, we can see that the price that we must pay to have an error bound in L_p-norm is the appearance of the concentrability coefficients $\mathcal{C}_q^{1,\infty,0}$ and $\mathcal{C}_q^{0,\infty,0}$. Furthermore, it is easy to see that the above bound is more general, i.e., by sending p to infinity, we recover the max-norm bound of Equation 22.

Remark 9 *We can balance the influence of the concentrability coefficients (the bigger the q, the higher the influence) and the difficulty of controlling the errors (the bigger the q', the greater the difficulty in controlling the $L_{pq'}$-norms) by tuning the parameters q and q', given that $\frac{1}{q} + \frac{1}{q'} = 1$. This potential leverage is an improvement over the existing bounds and concentrability results that only consider specific values of these two parameters: $q = \infty$ and $q' = 1$ in Munos (2007) and Munos and Szepesvári (2008), and $q = q' = 2$ in Farahmand* et al. *(2010).*

Remark 10 *It is important to note that our loss bound for AMPI does not "directly" depend on m (although as we will discuss in the next section, it "indirectly" does through ϵ_k). For CBMPI, the parameter m controls the influence of the value function approximator, cancelling it out in the limit when m tends to infinity (see Equation 24). Assuming a fixed budget of sample transitions, increasing m reduces the number of rollouts used by the classifier, and thus, worsens its quality. In such a situation, m allows making a trade-off between the estimation error of the classifier and the overall value function approximation.*

The arguments we developed globally follow those originally developed for λ-policy iteration (Scherrer, 2013). With respect to that work, our proof is significantly simpler thanks to the use of the Γ^n notation (Definition 3) and the fact that the AMPI scheme is itself much simpler than λ-policy iteration. Moreover, the results are deeper since we consider a possible error in the greedy step and more general concentration coefficients. Canbolat and Rothblum (2012) recently (and independently) developed an analysis of an approximate form of MPI. While Canbolat and Rothblum (2012) only consider the error in the greedy step, our work is more general since it takes into account both this error and the error in the value update. Note that it is required to consider both sources of error for the analysis of CBMPI. Moreover, Canbolat and Rothblum (2012) provide bounds when the errors are controlled in max-norm, while we consider the more general L_p-norm. At a more technical level, Theorem 2 in Canbolat and Rothblum (2012) bounds the norm of the distance $v_* - v_k$, while we bound the loss $v_* - v_{\pi_k}$. Finally, if we derive a bound on the loss (using *e.g.*, Theorem 1 in Canbolat and Rothblum 2012), this leads to a bound on the loss that is looser than ours. In particular, this does not allow to recover the standard bounds for AVI and API, as we may obtain here (in Equation 22).

The results that we just stated (Theorem 7 and 8) can be read as follows: if one can control the errors ϵ_k and ϵ'_k in L_p-norm, then the performance loss is also controlled.

The main limitation of this result is that in general, even if there is no sampling noise (*i.e.*, $N = \infty$ for all the algorithms and $M = \infty$ for AMPI-V), the error ϵ_k of the *evaluation step* may grow arbitrarily and make the algorithm diverge. The fundamental reason is that the composition of the approximation and the Bellman operator T_π is not necessarily contracting. Since the former is contracting with respect to the μ-norm, another reason for this issue is that T_π is in general not contracting for that norm. A simple well-known pathological example is due to Tsitsiklis and Van Roy (1997) and involves a two-state uncontrolled MDP and a linear projection onto a 1-dimensional space (that contains the real value function). Increasing the parameter m of the algorithm makes the operator $(T_\pi)^m$ used in Equation 19 more contracting and can in principle address this issue. For instance, if we consider that we have a state space of finite size $|\mathcal{S}|$, and take the uniform distribution μ, it can be easily seen that for any v and v', we have

$$\begin{aligned}\|(T_\pi)^m v - (T_\pi)^m v'\|_{2,\mu} &= \gamma^m \|(P_\pi)^m (v - v')\|_{2,\mu} \\ &\leq \gamma^m \|(P_\pi)^m\|_{2,\mu} \|v - v'\|_{2,\mu} \\ &\leq \gamma^m \sqrt{|\mathcal{S}|} \|v - v'\|_{2,\mu}.\end{aligned}$$

In other words, $(T_\pi)^m$ is contracting w.r.t. the μ-weighted norm as soon as $m > \frac{\log |\mathcal{S}|}{2 \log \frac{1}{\gamma}}$. In particular, it is sufficient for m to be exponentially smaller than the size of the state space in order to solve this potential divergence problem.

5. Finite-Sample Analysis of the Algorithms

In this section, we first show how the error terms ϵ_k and ϵ'_k appeared in Theorem 8 (Equations 23 and 24) can be bounded in each of the three proposed algorithms, and then use the obtained results and derive finite-sample performance bounds for these algorithms. We first bound the *evaluation step error* ϵ_k. In AMPI-V and CBMPI, the evaluation step at each iteration k is a regression problem with the target $(T_{\pi_k})^m v_{k-1}$ and a training set of the form $\left\{\left(s^{(i)}, \widehat{v}_k(s^{(i)})\right)\right\}_{i=1}^N$ in which the states $s^{(i)}$ are i.i.d. samples from the distribution μ and $\widehat{v}_k(s^{(i)})$'s are unbiased estimates of the target computed using Equation 5. The situation is the same for AMPI-Q, except everything is in terms of action-value function Q_k instead of value function v_k. Therefore, in the following we only show how to bound ϵ_k in AMPI-V and CBMPI, the extension to AMPI-Q is straightforward.

We may use linear or non-linear function space $\mathcal{F}$ to approximate $(T_{\pi_k})^m v_{k-1}$. Here we consider a linear architecture with parameters $\alpha \in \mathbb{R}^d$ and bounded (by L) basis functions $\{\varphi_j\}_{j=1}^d$, $\|\varphi_j\|_\infty \leq L$. We denote by $\phi : \mathcal{X} \to \mathbb{R}^d$, $\phi(\cdot) = \left(\varphi_1(\cdot), \ldots, \varphi_d(\cdot)\right)^\top$ the feature vector, and by $\mathcal{F}$ the linear function space spanned by the features φ_j, *i.e.*, $\mathcal{F} = \{f_\alpha(\cdot) = \phi(\cdot)^\top \alpha : \alpha \in \mathbb{R}^d\}$. Now if we define v_k as the truncation (by $V_{\max}$) of the solution of the above linear regression problem, we may bound the *evaluation step error* ϵ_k using the following lemma.

Lemma 11 (Evaluation step error) *Consider the linear regression setting described above, then we have*

$$\|\epsilon_k\|_{2,\mu} \leq 4 \inf_{f \in \mathcal{F}} \|(T_{\pi_k})^m v_{k-1} - f\|_{2,\mu} + e_1(N, \delta) + e_2(N, \delta),$$

with probability at least $1-\delta$*, where*

$$e_1(N,\delta) = 32 V_{\max}\sqrt{\frac{2}{N}\log\Big(\frac{27(12e^2N)^{2(d+1)}}{\delta}\Big)}\,,$$
$$e_2(N,\delta) = 24\Big(V_{\max} + \|\alpha_*\|_2 \cdot \sup_x \|\phi(x)\|_2\Big)\sqrt{\frac{2}{N}\log\frac{9}{\delta}}\,,$$

and α_* *is such that* f_{α_*} *is the best approximation (w.r.t.* μ*) of the target function* $(T_{\pi_k})^m v_{k-1}$ *in* $\mathcal{F}$*.*

Proof See Appendix D. ■

After we showed how to bound the *evaluation step error* ϵ_k for the proposed algorithms, we now turn our attention to bounding the *greedy step error* ϵ'_k, that contrary to the evaluation step error, varies more significantly across the algorithms. While the greedy step error equals to zero in AMPI-Q, it is based on sampling in AMPI-V, and depends on a classifier in CBMPI. To bound the *greedy step error* in AMPI-V and CBMPI, we assume that the action space $\mathcal{A}$ contains only two actions, i.e., $|\mathcal{A}| = 2$. The extension to more than two actions is straightforward along the same line of analysis as in Section 6 of Lazaric *et al.* (2010b). The main difference w.r.t. the two action case is that the VC-dimension of the policy space is replaced with its Natarajan dimension. We begin with AMPI-V.

Lemma 12 (Greedy step error of AMPI-V) *Let* μ *be a distribution over the state space* $\mathcal{S}$ *and* N *be the number of states in the rollout set* $\mathcal{D}_k$ *drawn i.i.d. from* μ*. For each state* $s \in \mathcal{D}_k$ *and each action* $a \in \mathcal{A}$*, we sample* M *states resulted from taking action* a *in state* s*. Let* h *be the VC-dimension of the policy space obtained by Equation 4 from the truncation (by* $V_{\max}$*) of the function space* $\mathcal{F}$*. For any* $\delta > 0$*, the greedy step error* ϵ'_k *in the AMPI-V algorithm is bounded as*

$$||\epsilon'_k(s)||_{1,\mu} \le e'_3(N,\delta) + e'_4(M,N,\delta) + e'_5(M,N,\delta),$$

with probability at least $1-\delta$*, with*

$$e'_3(N,\delta) = 16 V_{\max}\sqrt{\frac{2}{N}(h\log\frac{eN}{h} + \log\frac{24}{\delta})}\,,$$
$$e'_4(N,M,\delta) = 8 V_{\max}\sqrt{\frac{2}{MN}\big(h\log\frac{eMN}{h} + \log\frac{24}{\delta}\big)}\,, \qquad e'_5(M,N,\delta) = V_{\max}\sqrt{\frac{2\log(3N/\delta)}{M}}\,.$$

Proof See Appendix E. ■

We now show how to bound ϵ'_k in CBMPI. From the definitions of ϵ'_k (Equation 20) and $\mathcal{L}^{\Pi}_k(\mu;\pi)$ (Equation 15), it is easy to see that $\|\epsilon'_k\|_{1,\mu} = \mathcal{L}^{\Pi}_{k-1}(\mu;\pi_k)$. This is because

$$\begin{aligned}\epsilon'_k(s) &= \max_{a\in\mathcal{A}} \big[T_a(T_{\pi_{k-1}})^m v_{k-2}\big](s) - \big[T_{\pi_k}(T_{\pi_{k-1}})^m v_{k-2}\big](s) && \text{(see Equation 13)}\\ &= \max_{a\in\mathcal{A}} Q_{k-1}(s,a) - Q_{k-1}\big(s,\pi_k(s)\big). && \text{(see Equations 14 and 15)}\end{aligned}$$

Lemma 13 (Greedy step error of CBMPI) *Let the policy space Π defined by the classifier have finite VC-dimension $h = VC(\Pi) < \infty$, and μ be a distribution over the state space $\mathcal{S}$. Let N' be the number of states in $\mathcal{D}'_{k-1}$ drawn i.i.d. from μ, M be the number of rollouts per state-action pair used in the estimation of $\widehat{Q}_{k-1}$, and $\pi_k = \operatorname{argmin}_{\pi\in\Pi} \widehat{\mathcal{L}}^{\Pi}_{k-1}(\widehat{\mu}, \pi)$ be the policy computed at iteration $k-1$ of CBMPI. Then, for any $\delta > 0$, we have*

$$\|\epsilon'_k\|_{1,\mu} = \mathcal{L}^{\Pi}_{k-1}(\mu; \pi_k) \leq \inf_{\pi\in\Pi} \mathcal{L}^{\Pi}_{k-1}(\mu; \pi) + 2\big(e'_1(N', \delta) + e'_2(N', M, \delta)\big),$$

with probability at least $1-\delta$, where

$$e'_1(N', \delta) = 16 Q_{\max} \sqrt{\frac{2}{N'}\big(h \log \frac{eN'}{h} + \log \frac{32}{\delta}\big)}\ ,$$
$$e'_2(N', M, \delta) = 8 Q_{\max} \sqrt{\frac{2}{MN'}\big(h \log \frac{eMN'}{h} + \log \frac{32}{\delta}\big)}\ .$$

Proof See Appendix F. ■

From Lemma 11, we have a bound on $\|\epsilon_k\|_{2,\mu}$ for all the three algorithms. Since $\|\epsilon_k\|_{1,\mu} \leq \|\epsilon_k\|_{2,\mu}$, we also have a bound on $\|\epsilon_k\|_{1,\mu}$ for all the algorithms. On the other hand, from Lemmas 12 and 13, we have a bound on $\|\epsilon'_k\|_{1,\mu}$ for the AMPI-V and CMBPI algorithms. This means that for AMPI-V, AMPI-Q ($\epsilon'_k = 0$ for this algorithm), and CBMPI, we can control the right hand side of Equations 23 and 24 in L_1-norm, which in the context of Theorem 8 means $p = 1$, $q' = 1$, and $q = \infty$. This leads to the main result of this section, finite-sample performance bounds for the three proposed algorithms.

Theorem 14 *Let*

$$d' = \sup_{g\in\mathcal{F}, \pi'\in\Pi} \inf_{\pi\in\Pi} \mathcal{L}^{\Pi}_{\pi', g}(\mu; \pi) \qquad \text{and} \qquad d_m = \sup_{g\in\mathcal{F}, \pi\in\Pi} \inf_{f\in\mathcal{F}} \|(T_\pi)^m g - f\|_{2,\mu}$$

where $\mathcal{F}$ is the function space used by the algorithms and Π is the policy space used by CBMPI with the VC-dimension h. With the notations of Theorem 8 and Lemmas 11-13, after k iterations, and with probability $1-\delta$, the expected losses $\mathbb{E}_\rho[l_k] = \|l_k\|_{1,\rho}$ of the proposed AMPI algorithms satisfy:[6]

AMPI-V:
$$\|l_k\|_{1,\rho} \leq \frac{2(\gamma - \gamma^k)\mathcal{C}^{1,k,0}_\infty}{(1-\gamma)^2}\left(d_m + e_1(N, \frac{\delta}{k}) + e_2(N, \frac{\delta}{k})\right) + \frac{(1-\gamma^k)\mathcal{C}^{0,k,0}_\infty}{(1-\gamma)^2}\left(e'_3(N, \frac{\delta}{k}) + e'_4(N, M, \frac{\delta}{k}) + e'_5(N, M, \frac{\delta}{k})\right) + g(k),$$

AMPI-Q:
$$\|l_k\|_{1,\rho} \leq \frac{2(\gamma - \gamma^k)\mathcal{C}^{1,k,0}_\infty}{(1-\gamma)^2}\left(d_m + e_1(N, \frac{\delta}{k}) + e_2(N, \frac{\delta}{k})\right) + g(k),$$

6. Note that the bounds of AMPI-V and AMPI-Q may also be written with $(p = 2, q' = 1, q = \infty)$, and $(p = 1, q' = 2, q = 2)$.

CBMPI:
$$\|l_k\|_{1,\rho} \leq \frac{2\gamma^m(\gamma-\gamma^{k-1})\mathcal{C}_\infty^{2,k,m}}{(1-\gamma)^2}\left(d_m + e_1(N,\frac{\delta}{2k}) + e_2(N,\frac{\delta}{2k})\right) + \frac{(1-\gamma^k)\mathcal{C}_\infty^{1,k,0}}{(1-\gamma)^2}\left(d' + e'_1(N',\frac{\delta}{2k}) + e'_2(N',M,\frac{\delta}{2k})\right) + g(k).$$

Remark 15 *Assume that we run AMPI-Q with a total fixed budget B that is equally divided between the K iterations.*[7] *Recall from Theorem 8 that $g(k) = \gamma^k \mathcal{C}_q^{k,k+1,0} C_0$, where $C_0 = \min\left(\|d_0\|_{pq',\mu}, \|b_0\|_{pq',\mu}\right) \leq V_{\max}$ measures the quality of the initial value/policy pair. Then, up to constants and logarithmic factors, one can see that the bound has the form*

$$\|l_k\|_{1,\mu} \leq O\left(d_m + \sqrt{\frac{K}{B}} + \gamma^K C_0\right).$$

We deduce that the best choice for the number of iterations K can be obtained as a compromise between the quality of the initial value/policy pair and the estimation errors of the value estimation step.

Remark 16 *The CBMPI bound in Theorem 14 allows to turn the* qualitative *Remark 10 into a* quantitative *one. Assume that we have a fixed budget per iteration $B = Nm + N'M|\mathcal{A}|(m+1)$ that is equally divided over the classifier and regressor. Note that the budget is measured in terms of the number of calls to the generative model. Then up to constants and logarithmic factors, the bound has the form*

$$\|l_k\|_{1,\mu} \leq O\left(\gamma^m\left(d_m + \sqrt{\frac{m}{B}}\right) + d' + \sqrt{\frac{|A|m}{B}}(\sqrt{n}+\sqrt{M})\right).$$

This shows a trade-off in tuning the parameter m: a large value of m makes the influence (in the final error) of the regressor's error (both approximation and estimation errors) smaller, and at the same time the influence of the estimation error of the classifier larger.

6. Experimental Results

The main objective of this section is to present experiments for the new algorithm that we think is the most interesting, CBMPI, but we shall also illustrate AMPI-Q (we do not illustrate AMPI-V that is close to AMPI-Q but significantly less efficient to implement). We consider two different domains: **1)** the *mountain car* problem and **2)** the more challenging game of *Tetris*. In several experiments, we compare the performance of CBMPI with the DPI algorithm (Lazaric *et al.*, 2010a), which is basically CBMPI without value function approximation.[8] Note that comparing DPI and CBMPI allows us to highlight the role of the value function approximation.

As discussed in Remark 10, the parameter m in CBMPI balances between the errors in evaluating the value function and the policy. The value function approximation error

7. Similar reasonings can be done for AMPI-V and CBMPI, we selected AMPI-Q for the sake of simplicity. Furthermore, one could easily relax the assumption that the budget is *equally* divided by using Theorem 7.
8. DPI, as it is presented by Lazaric *et al.* (2010a), uses infinitely long rollouts and is thus equivalent to CBMPI with $m = \infty$. In practice, implementations of DPI use rollouts that are truncated after some horizon H, and is then equivalent to CBMPI with $m = H$ and $v_k = 0$ for all the iterations k.

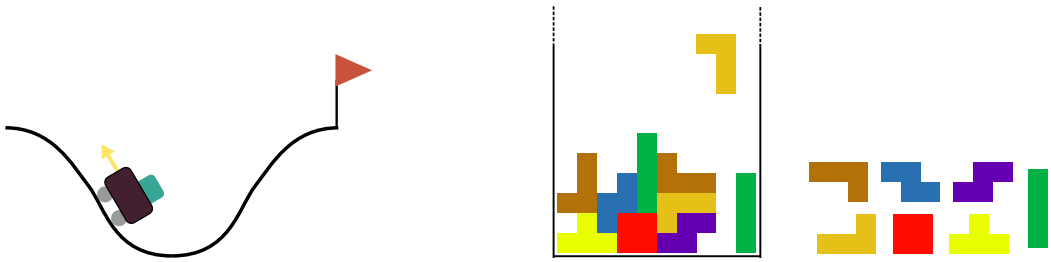


Figure 4: ***(Left)*** The Mountain Car (MC) problem in which the car needs to learn to oscillate back and forth in order to build up enough inertia to reach the top of the one-dimensional hill. ***(Right)*** A screen-shot of the game of Tetris and the seven pieces (shapes) used in the game.

tends to zero for large values of m. Although this would suggest to have large values for m, as mentioned in Remark 16, the size of the rollout sets $\mathcal{D}$ and $\mathcal{D}'$ would correspondingly decreases as $N = O(B/m)$ and $N' = O(B/m)$, thus decreasing the accuracy of both the regressor and classifier. This leads to a trade-off between long rollouts and the number of states in the rollout sets. The solution to this trade-off strictly depends on the capacity of the value function space $\mathcal{F}$. A rich value function space would lead to solve the trade-off for small values of m. On the other hand, when the value function space is poor, or, as in the case of DPI, when there is no value function, m should be selected in a way to guarantee large enough rollout sets (parameters N and N'), and at the same time, a sufficient number of rollouts (parameter M).

One of the objectives of our experiments is to show the role of these parameters in the performance of CBMPI. However, since we almost always obtained our best results with $M = 1$, we only focus on the parameters m and N in our experiments. Moreover, as mentioned in Footnote 3, we implement a more sample-efficient version of CBMPI by reusing the rollouts generated for the classifier in the regressor. More precisely, at each iteration k, for each state $s^{(i)} \in \mathcal{D}'_k$ and each action $a \in \mathcal{A}$, we generate one rollout of length $m+1$, i.e., $\big(s^{(i)}, a, r_0^{(i)}, s_1^{(i)}, a_1^{(i)}, \ldots, a_m^{(i)}, r_m^{(i)}, s_{m+1}^{(i)}\big)$. We then take the rollout of action $\pi_k(s^{(i)})$, select its last m steps, i.e., $\big(s_1^{(i)}, a_1^{(i)}, \ldots, a_m^{(i)}, r_m^{(i)}, s_{m+1}^{(i)}\big)$ (note that all the actions here have been taken according to the current policy π_k), use it to estimate the value function $\widehat{v}_k(s_1^{(i)})$, and add it to the training set of the regressor. This process guarantees to have $N = N'$.

In each experiment, we run the algorithms with the same budget B per iteration. The budget B is the number of next state samples generated by the generative model of the system at each iteration. In DPI and CBMPI, we generate a rollout of length $m+1$ for each state in $\mathcal{D}'$ and each action in $\mathcal{A}$, so, $B = (m+1)N|\mathcal{A}|$. In AMPI-Q, we generate one rollout of length m for each state-action pair in $\mathcal{D}$, and thus, $B = mN$.

6.1 Mountain Car

Mountain Car (MC) is the problem of driving a car up to the top of a one-dimensional hill (see Figure 4). The car is not powerful enough to accelerate directly up the hill, and

thus, it must learn to oscillate back and forth to build up enough inertia. There are three possible actions: forward (+1), reverse (−1), and stay (0). The reward is −1 for all the states but the goal state at the top of the hill, where the episode ends with a reward 0. The discount factor is set to $\gamma = 0.99$. Each state s consists of the pair $(x_s, \dot{x}_s)$, where x_s is the position of the car and $\dot{x}_s$ is its velocity. We use the formulation described in Dimitrakakis and Lagoudakis (2008) with uniform noise in $[-0.2, 0.2]$ added to the actions.

In this section, we report the empirical evaluation of CBMPI and AMPI-Q and compare it to DPI and LSPI (Lagoudakis and Parr, 2003a) in the MC problem. In our experiments, we show that CBMPI, by combining policy and value function approximation, can improve over AMPI-Q, DPI, and LSPI.

6.1.1 Experimental Setup

The value function is approximated using a linear space spanned by a set of radial basis functions (RBFs) evenly distributed over the state space. More precisely, we uniformly divide the 2-dimensional state space into a number of regions and place a Gaussian function at the center of each of them. We set the standard deviation of the Gaussian functions to the width of a region. The function space to approximate the action-value function in LSPI is obtained by replicating the state-features for each action. We run LSPI off-policy (i.e., samples are collected once and reused through the iterations of the algorithm).

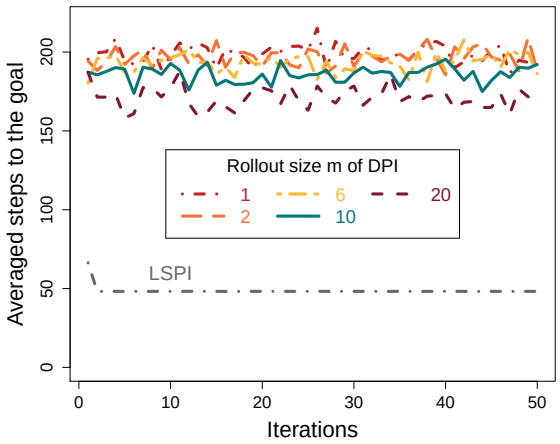
The policy space Π (classifier) is defined by a regularized support vector classifier (C-SVC) using the LIBSVM implementation by Chang and Lin (2011). We use the RBF kernel $\exp(-|u - v|^2)$ and set the cost parameter $C = 1000$. We minimize the classification error instead of directly solving the cost-sensitive multi-class classification step as in Figure 3. In fact, the classification error is an upper-bound on the empirical error defined by Equation 17. Finally, the rollout set is sampled uniformly over the state space.

In our MC experiments, the policies learned by the algorithms are evaluated by the number of steps-to-go (average number of steps to reach the goal with a maximum of 300) averaged over $4,000$ independent trials. More precisely, we define the possible starting configurations (positions and velocities) of the car by placing a 20×20 uniform grid over the state space, and run the policy 6 times from each possible initial configuration. The performance of each algorithm is represented by a learning curve whose value at each iteration is the average number of steps-to-go of the policies learned by the algorithm at that iteration in $1,000$ separate runs of the algorithm.

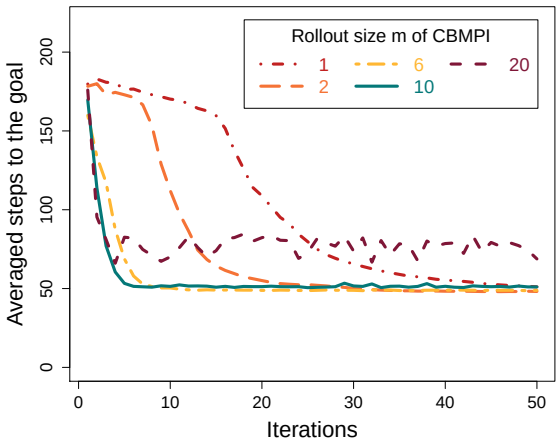
We tested the performance of DPI, CBMPI, and AMPI-Q on a wide range of parameters (m, M, N), but only report their performance for the best choice of M (as mentioned earlier, $M = 1$ was the best choice in all the experiments) and different values of m.

6.1.2 Experimental Results

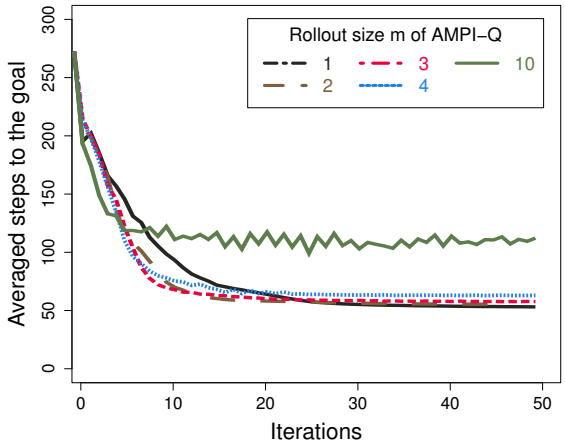
Figure 5 shows the learning curves of DPI, CBMPI, AMPI-Q, and LSPI algorithms with budget $B = 4,000$ per iteration and the function space $\mathcal{F}$ composed of a 3×3 RBF grid. We notice from the results that this space is rich enough to provide a good approximation for the value function components (e.g., in CBMPI, for $(T_\pi)^m v_{k-1}$ defined by Equation 19). Therefore, LSPI and DPI obtain the best and worst results with about 50 and 160 steps to reach the goal, respectively. The best DPI results are obtained with the large value of



(a) Performance of DPI (for different values of m) and LSPI.



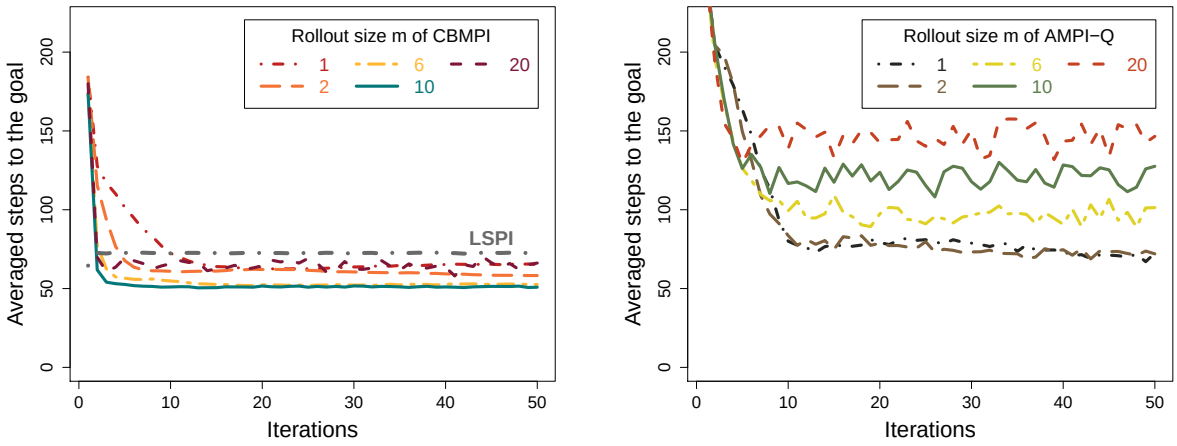
(b) Performance of CBMPI for different values of m.



(c) Performance of AMPI-Q for different values of m.

Figure 5: Performance of the policies learned by **(a)** DPI and LSPI, **(b)** CBMPI, and **(c)** AMPI-Q algorithms in the Mountain Car (MC) problem, when we use a 3×3 RBF grid to approximate the value function. The results are averaged over $1,000$ runs. The total budget B is set to $4,000$ per iteration.

$m = 20$. DPI performs better for large values of m because the reward function is constant everywhere except at the goal, and thus, a DPI rollout is only *informative* if it reaches the goal. We also report the performance of CBMPI and AMPI-Q for different values of m. The value function approximation is very accurate, and thus, CBMPI and AMPI-Q achieve performance similar to LSPI for $m < 20$. However when m is large ($m = 20$), the performance of these algorithms is worse, because in this case, the rollout set does not have enough elements (N small) to learn the greedy policy and value function well. Note that as we increase m (up to $m = 10$), CBMPI and AMPI-Q converge faster to a good policy.



(a) Performance of CBMPI (for different values of m) and LSPI.

(b) Performance of AMPI-Q for different values of m.

Figure 6: Performance of the policies learned by **(a)** CBMPI and LSPI and **(b)** AMPI-Q algorithms in the Mountain Car (MC) problem, when we use a 2×2 RBF grid to approximate the value function. The results are averaged over $1,000$ runs. The total budget B is set to $4,000$ per iteration.

Although this experiment shows that the use of a critic in CBMPI compensates for the truncation of the rollouts (CBMPI performs better than DPI), most of this advantage is due to the richness of the function space $\mathcal{F}$ (LSPI and AMPI-Q perform as well as CBMPI – LSPI even converges faster). Therefore, it seems that it would be more efficient to use LSPI instead of CBMPI in this case.

In the next experiment, we study the performance of these algorithms when the function space $\mathcal{F}$ is less rich, composed of a 2×2 RBF grid. The results are reported in Figure 6. Now, the performance of LSPI and AMPI-Q (for the best value of $m = 1$) degrades to 75 and 70 steps, respectively. Although $\mathcal{F}$ is not rich, it still helps CBMPI to outperform DPI. We notice the effect of (a weaker) $\mathcal{F}$ in CBMPI when we observe that it no longer converges to its best performance (about 50 steps) for small values of $m = 1$ and $m = 2$. Note that CMBPI outperforms all the other algorithms for $m = 10$ (and even for $m = 6$), while still has a sub-optimal performance for $m = 20$, mainly due to the fact that the rollout set would be too small in this case.

6.2 Tetris

Tetris is a popular video game created by Alexey Pajitnov in 1985. The game is played on a grid originally composed of 20 rows and 10 columns, where pieces of 7 different shapes fall from the top (see Figure 4). The player has to choose where to place each falling piece by moving it horizontally and rotating it. When a row is filled, it is removed and all the cells above it move one line down. The goal is to remove as many rows as possible before the game is over, i.e., when there is no space available at the top of the grid for the new piece. This game constitutes an interesting optimization benchmark in which the goal is to

find a controller (policy) that maximizes the average (over multiple games) number of lines removed in a game (score).[9] This optimization problem is known to be computationally hard. It contains a huge number of board configurations (about $2^{200} \simeq 1.6 \times 10^{60}$), and even in the case that the sequence of pieces is known in advance, finding the strategy to maximize the score is a NP hard problem (Demaine *et al.*, 2003). Here, we consider the variation of the game in which the player only knows the current falling piece and none of the next several coming pieces.

Approximate dynamic programming (ADP) and reinforcement learning (RL) algorithms including approximate value iteration (Tsitsiklis and Van Roy, 1996), λ-policy iteration (λ-PI) (Bertsekas and Ioffe, 1996; Scherrer, 2013), linear programming (Farias and Van Roy, 2006), and natural policy gradient (Kakade, 2002; Furmston and Barber, 2012) have been applied to this very setting. These methods formulate Tetris as a MDP (with discount factor $\gamma = 1$) in which the state is defined by the current board configuration plus the falling piece, the actions are the possible orientations of the piece and the possible locations that it can be placed on the board,[10] and the reward is defined such that maximizing the expected sum of rewards from each state coincides with maximizing the score from that state. Since the state space is large in Tetris, these methods use value function approximation schemes (often linear approximation) and try to tune the value function parameters (weights) from game simulations. Despite a long history, ADP/RL algorithms, that have been (almost) entirely based on approximating the value function, have not been successful in finding good policies in Tetris. On the other hand, methods that search directly in the space of policies by learning the policy parameters using black-box optimization, such as the cross entropy (CE) method (Rubinstein and Kroese, 2004), have achieved the best reported results in this game (see e.g., Szita and Lőrincz 2006; Thiery and Scherrer 2009b). This makes us conjecture that Tetris is a game in which good policies are easier to represent, and thus to learn, than their corresponding value functions. So, in order to obtain a good performance with ADP in Tetris, we should use those ADP algorithms that search in a policy space, like CBMPI and DPI, instead of the more traditional ones that search in a value function space.

In this section, we evaluate the performance of CBMPI in Tetris and compare it with DPI, λ-PI, and CE. In these experiments, we show that CBMPI improves over all the previously reported ADP results. Moreover, it obtains the best results reported in the literature for Tetris in both small 10×10 and large 10×20 boards. Although the CBMPI's results are similar to those achieved by the CE method in the large board, it uses considerably fewer samples (call to the generative model of the game) than CE.

6.2.1 Experimental Setup

In this section, we briefly describe the algorithms used in our experiments: the cross entropy (CE) method, our particular implementation of CBMPI, and its slight variation DPI. We refer the readers to Scherrer (2013) for λ-PI. We begin by defining some terms and notations. A state s in Tetris consists of two components: the description of the board b and the type of the falling piece p. All controllers rely on an evaluation function that gives a value to each

9. Note that this number is finite because it was shown that Tetris is a game that ends with probability one (Burgiel, 1997).

10. The total number of actions at a state depends on the shape of the falling piece, with the maximum of 32 actions in a state, i.e., $|\mathcal{A}| \leq 32$.

possible action at a given state. Then, the controller chooses the action with the highest value. In ADP, algorithms aim at tuning the weights such that the evaluation function approximates well the value function, which coincides with the optimal expected future score from each state. Since the total number of states is large in Tetris, the evaluation function f is usually defined as a linear combination of a set of features ϕ, i.e., $f(\cdot) = \phi(\cdot)^\top \theta$. Alternatively, we can think of the parameter vector θ as a policy (controller) whose performance is specified by the corresponding evaluation function $f(\cdot) = \phi(\cdot)^\top \theta$. The features used in Tetris for a state-action pair (s, a) may depend on the description of the board b' resulting from taking action a in state s, e.g., the maximum height of b'. Computing such features requires to exploit the knowledge of the game's dynamics (this dynamics is indeed known for tetris). We consider the following sets of features, plus a constant offset feature:[11]

(i) Bertsekas Features: First introduced by Bertsekas and Tsitsiklis (1996), this set of 22 features has been mainly used in the ADP/RL community and consists of: *the number of holes in the board*, *the height of each column*, *the difference in height between two consecutive columns*, and *the maximum height of the board*.

(ii) Dellacherie-Thiery (D-T) Features: This set consists of the six features of Dellacherie (Fahey, 2003), i.e., *the landing height of the falling piece*, *the number of eroded piece cells*, *the row transitions*, *the column transitions*, *the number of holes*, and *the number of board wells*; plus 3 additional features proposed in Thiery and Scherrer (2009b), i.e., *the hole depth*, *the number of rows with holes*, and *the pattern diversity feature*. Note that the best policies reported in the literature have been learned using this set of features.

(iii) RBF Height Features: These new 5 features are defined as $\exp(\frac{-|c-ih/4|^2}{2(h/5)^2})$, $i = 0, \ldots, 4$, where c is the average height of the columns and $h = 10$ or 20 is the total number of rows in the board.

The Cross Entropy (CE) Method: CE (Rubinstein and Kroese, 2004) is an iterative method whose goal is to optimize a function f parameterized by a vector $\theta \in \Theta$ by direct search in the parameter space Θ. Figure 7 contains the pseudo-code of the CE algorithm used in our experiments (Szita and Lőrincz, 2006; Thiery and Scherrer, 2009b). At each iteration k, we sample n parameter vectors $\{\theta_i\}_{i=1}^n$ from a multivariate Gaussian distribution with diagonal covariance matrix $\mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$. At the beginning, the parameters of this Gaussian have been set to cover a wide region of Θ. For each parameter θ_i, we play G games and calculate the average number of rows removed by this controller (an estimate of the expected score). We then select $\lfloor \zeta n \rfloor$ of these parameters with the highest score, $\theta'_1, \ldots, \theta'_{\lfloor \zeta n \rfloor}$, and use them to update the mean $\boldsymbol{\mu}$ and variance $\mathrm{diag}(\boldsymbol{\sigma}^2)$ of the Gaussian distribution, as shown in Figure 7. This updated Gaussian is used to sample the n parameters at the next iteration. The goal of this update is to sample more parameters from the promising parts of Θ at the next iteration, and hopefully converge to a good

11. For a precise definition of the features, see Thiery and Scherrer (2009a) or the documentation of their code (Thiery and Scherrer, 2010b). Note that the constant offset feature only plays a role in value function approximation, and has no effect in modeling polices.

Input: parameter space Θ, number of parameter vectors n, proportion $\zeta \leq 1$, noise η
Initialize: Set the mean and variance parameters $\boldsymbol{\mu} = (0, 0, \ldots, 0)$ and $\boldsymbol{\sigma}^2 = (100, 100, \ldots, 100)$
for $k = 1, 2, \ldots$ **do**
  Generate a random sample of n parameter vectors $\{\theta_i\}_{i=1}^{n} \sim \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$
  For each θ_i, play G games and calculate the average number of rows removed (score) by the controller
  Select $\lfloor \zeta n \rfloor$ parameters with the highest score $\theta'_1, \ldots, \theta'_{\lfloor \zeta n \rfloor}$
  Update $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$: $\mu(j) = \frac{1}{\lfloor \zeta n \rfloor} \sum_{i=1}^{\lfloor \zeta n \rfloor} \theta'_i(j)$ and $\sigma^2(j) = \frac{1}{\lfloor \zeta n \rfloor} \sum_{i=1}^{\lfloor \zeta n \rfloor} [\theta'_i(j) - \mu(j)]^2 + \eta$
end for

Figure 7: The pseudo-code of the cross-entropy (CE) method used in our experiments.

maximum of f. In our experiments, in the pseudo-code of Figure 7, we set $\zeta = 0.1$ and $\eta = 4$, the best parameters reported in Thiery and Scherrer (2009b). We also set $n = 1,000$ and $G = 10$ in the small board (10×10) and $n = 100$ and $G = 1$ in the large board (10×20).

Our Implementation of CBMPI (DPI): We use the algorithm whose pseudo-code is shown in Figure 3. We sampled states from the trajectories generated by a good policy for Tetris, namely the DU controller obtained by Thiery and Scherrer (2009b). Since this policy is good, this set is biased towards boards with small height. The rollout set is then obtained by subsampling this set so that the board height distribution is more uniform. We noticed from our experiments that this subsampling significantly improves the performance. We now describe how we implement the regressor and the classifier.

- **Regressor:** We use linear function approximation for the value function, i.e., $\widehat{v}_k(s^{(i)}) = \phi(s^{(i)})\alpha$, where $\phi(\cdot)$ and α are the feature and weight vectors, and minimize the empirical error $\widehat{\mathcal{L}}_k^{\mathcal{F}}(\widehat{\mu}; v)$ using the standard least-squares method.
- **Classifier:** The training set of the classifier is of size N with $s^{(i)} \in \mathcal{D}'_k$ as input and $\big(\max_a \widehat{Q}_k(s^{(i)}, a) - \widehat{Q}_k(s^{(i)}, a_1), \ldots, \max_a \widehat{Q}_k(s^{(i)}, a) - \widehat{Q}_k(s^{(i)}, a_{|\mathcal{A}|})\big)$ as output. We use the policies of the form $\pi_\beta(s) = \mathrm{argmax}_a \, \psi(s, a)^\top \beta$, where ψ is the policy feature vector (possibly different from the value function feature vector ϕ) and $\beta \in \mathcal{B}$ is the policy parameter vector. We compute the next policy π_{k+1} by minimizing the empirical error $\widehat{\mathcal{L}}_k^{\Pi}(\widehat{\mu}; \pi_\beta)$, defined by (17), using the covariance matrix adaptation evolution strategy (CMA-ES) algorithm (Hansen and Ostermeier, 2001). In order to evaluate a policy $\beta \in \mathcal{B}$ in CMA-ES, we only need to compute $\widehat{\mathcal{L}}_k^{\Pi}(\widehat{\mu}; \pi_\beta)$, and given the training set, this procedure does not require further simulation of the game.

We set the initial value function parameter to $\alpha = (0, 0, \ldots, 0)$ and select the initial policy π_1 (policy parameter β) randomly. We also set the CMA-ES parameters (classifier parameters) to $\zeta = 0.5$, $\eta = 0$, and n equal to 15 times the number of features.

6.2.2 Experiments

In our Tetris experiments, the policies learned by the algorithms are evaluated by their score (average number of rows removed in a game started with an empty board) averaged

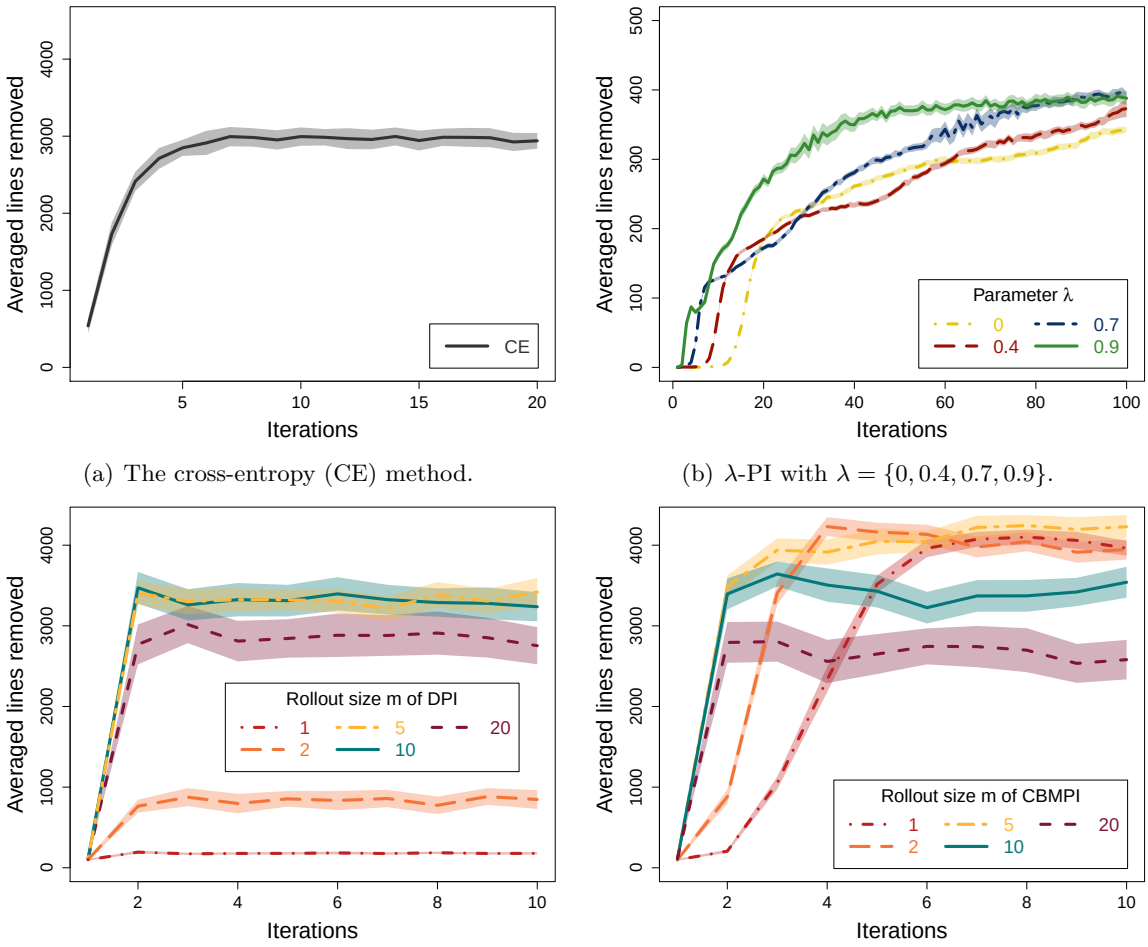
over 200 games in the small 10×10 board and over 20 games in the large 10×20 board (since the game takes much more time to complete in the large board). The performance of each algorithm is represented by a learning curve whose value at each iteration is the average score of the policies learned by the algorithm at that iteration in 100 separate runs of the algorithm. The curves are wrapped in their confidence intervals that are computed as three time the standard deviation of the estimation of the performance at each iteration. In addition to their score, we also evaluate the algorithms by the number of samples they use. In particular, we show that CBMPI/DPI use 6 times fewer samples than CE in the large board. As discussed in Section 6.2.1, this is due the fact that although the classifier in CBMPI/DPI uses a direct search in the space of policies (for the greedy policy), it evaluates each candidate policy using the empirical error of Equation 17, and thus, does not require any simulation of the game (other than those used to estimate the $\widehat{Q}_k$'s in its training set). In fact, the budget B of CBMPI/DPI is fixed in advance by the number of rollouts NM and the rollout's length m as $B = (m+1)NM|\mathcal{A}|$. On the contrary, CE evaluates a candidate policy by playing several games, a process that can be extremely costly (sample-wise), especially for good policies in the large board.

We first run the algorithms on the small board to study the role of their parameters and to select the best features and parameters, and then use the selected features and parameters and apply the algorithms to the large board. Finally, we compare the best policies found in our experiments with the best controllers reported in the literature (Tables 1 and 2).

Small (10×10) Board

Here we run the algorithms with two different feature sets: *Dellacherie-Thiery (D-T)* and *Bertsekas*, and report their results.

D-T Features: Figure 8 shows the learning curves of CE, λ-PI, DPI, and CBMPI. Here we use the D-T features for the evaluation function in CE, the policy in DPI and CBMPI, and the value function in λ-PI (in the last case we also add the constant offset feature). For the value function of CBMPI, we tried different choices of features and "D-T plus the 5 RBF features and constant offset" achieved the best performance (see Figure 8(d)). The budget of CBMPI and DPI is set to $B = 8,000,000$ per iteration. The CE method reaches the score $3,000$ after 10 iterations using an average budget $B = 65,000,000$. λ-PI with the best value of λ only manages to score 400. In Figure 8(c), we report the performance of DPI for different values of m. DPI achieves its best performance for $m = 5$ and $m = 10$ by removing $3,400$ lines on average. As explained in Section 6.1, having short rollouts ($m = 1$) in DPI leads to poor action-value estimates $\widehat{Q}$, while having too long rollouts ($m = 20$) decreases the size of the training set of the classifier N. CBMPI outperforms the other algorithms, including CE, by reaching the score of $4,200$ for $m = 5$. This value of $m = 5$ corresponds to $N = \frac{8000000}{(5+1)\times 32} \approx 42,000$. Note that unlike DPI, CBMPI achieves good performance with very short rollouts $m = 1$. This indicates that CBMPI is able to approximate the value function well, and as a result, build a more accurate training set for its classifier than DPI. Despite this improvement, the good results obtained by DPI in Tetris indicate that with small rollout horizons like $m = 5$, one already has fairly accurate action-value estimates in order to detect greedy actions accurately (at each iteration).



(a) The cross-entropy (CE) method.

(b) λ-PI with $\lambda = \{0, 0.4, 0.7, 0.9\}$.

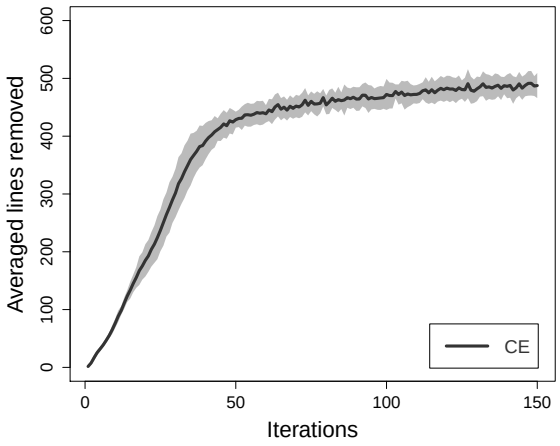
(c) DPI with budget $B = 8,000,000$ per iteration and $m = \{1, 2, 5, 10, 20\}$.

(d) CBMPI with budget $B = 8,000,000$ per iteration and $m = \{1, 2, 5, 10, 20\}$.

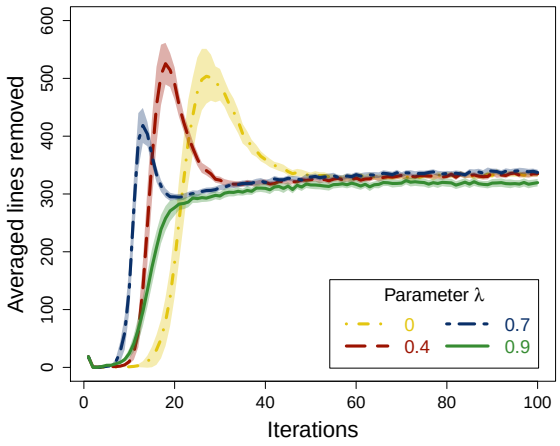
Figure 8: Learning curves of CE, λ-PI, DPI, and CBMPI using the 9 Dellacherie-Thiery (D-T) features on the small 10×10 board. The results are averaged over 100 runs of the algorithms.

Overall, the results of Figure 8 show that an ADP algorithm, namely CBMPI, outperforms the CE method using a similar budget (80 vs. 65 millions after 10 iterations). Note that CBMPI takes less iterations to converge than CE. More generally, Figure 8 confirms the superiority of the policy search and classification-based PI methods to value-function based ADP algorithms (λ-PI). This suggests that the D-T features are more suitable to represent policies than value functions in Tetris.

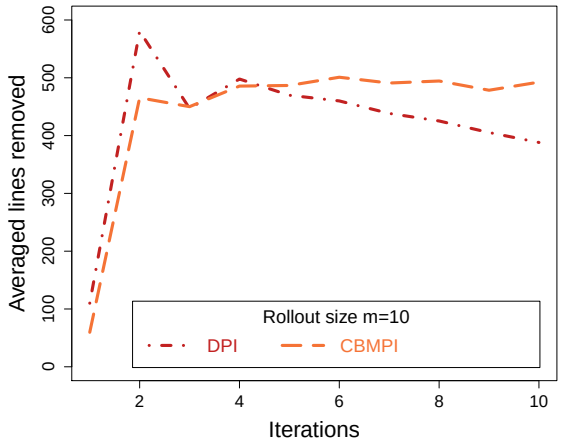
Bertsekas Features: Figures 9(a)-(c) show the performance of CE, λ-PI, DPI, and CBMPI. Here all the approximations in the algorithms are with the Bertsekas features plus constant offset. CE achieves the score 500 after about 60 iterations and outperforms λ-PI with score 350. It is clear that the Bertsekas features lead to much weaker results than those obtained



(a) The cross-entropy (CE) method.



(b) λ-PI with $\lambda = \{0, 0.4, 0.7, 0.9\}$.



(c) DPI (dash-dotted line) & CBMPI (dash line) with budget $B = 80,000,000$ per iteration and $m = 10$.

Figure 9: (a)-(c) Learning curves of CE, λ-PI, DPI, and CBMPI algorithms using the 22 Bertsekas features on the small 10×10 board.

by the D-T features (Figure 8) for all the algorithms. We may conclude then that the D-T features are more suitable than the Bertsekas features to represent both value functions and policies in Tetris. In DPI and CBMPI, we managed to obtain results similar to CE, only after multiplying the per iteration budget B used in the D-T experiments by 10. Indeed, CBMPI and DPI need more samples to solve the classification and regression problems in this 22-dimensional weight vector space than with the 9 D-T features. Moreover, in the classifier, the minimization of the empirical error through the CMA-ES method (see Equation 12) was converging most of the times to a local minimum. To solve this issue, we run multiple times the minimization problem with different starting points and small initial covariance matrices for the Gaussian distribution in order to force local exploration of different parts of the weight vector areas. However, CBMPI and CE require the same

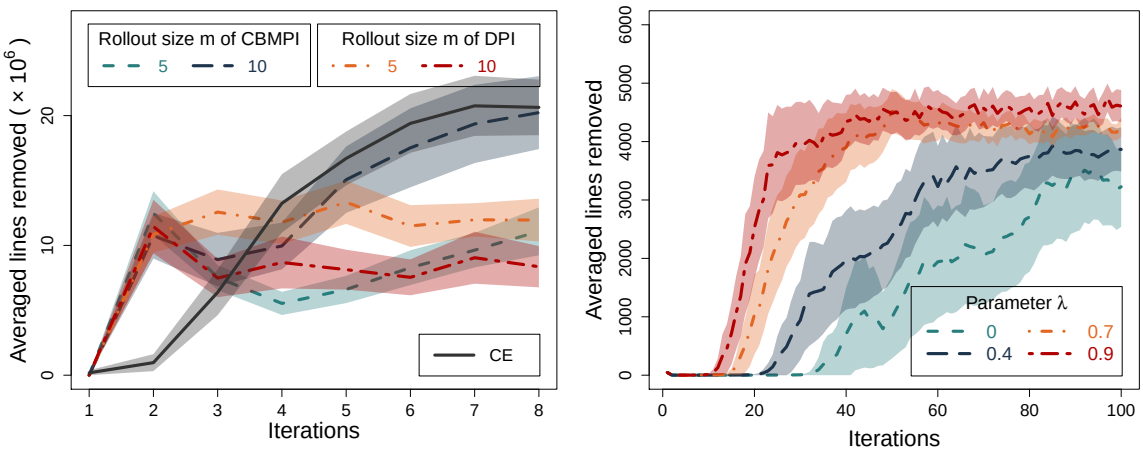


Figure 10: Learning curves of CBMPI, DPI and CE (left) using the 9 features listed in Table 2, and λ-PI (right) using the Bertsekas features (those for which λ-PI achieves here its best performance) on the large 10×20 board. The total budget B of CBMPI and DPI is set to 16,000,000 per iteration.

number of samples, $150,000,000$, to reach their best performance, after 2 and 60 iterations, respectively (see Figure 9). Note that DPI and CBMPI obtain the same performance, which means that the use of a value function approximation by CBMPI does not lead to a significant performance improvement over DPI. We tried several values of m in this setting among which $m = 10$ achieved the best performance for both DPI and CBMPI.

Large (10×20) Board

We now use the best parameters and features in the small board experiments, run CE, DPI, and CBMPI in the large board, and report their results in Figure 10 (left). We also report the results of λ-PI in the large board in Figure 10 (right). The per iteration budget of DPI and CBMPI is set to $B = 32,000,000$. While λ-PI with per iteration budget $100,000$, at its best, achieves the score of $2,500$, DPI and CBMPI, with $m = 5$ and $m = 10$, reach the scores of $12,000,000$ and $20,000,000$ after 3 and 8 iterations, respectively. CE matches the performances of CBMPI with the score of $20,000,000$ after 8 iterations, this is achieved with almost 6 times more samples: after 8 iterations, CBMPI and CE use $256,000,000$ and $1,700,000,000$ samples, respectively.

Comparison of the Best Policies

So far the reported scores for each algorithm was averaged over the policies learned in 100 separate runs. Here we select the best policies observed in all our experiments and compute their scores more accurately by averaging over $10,000$ games. We then compare these results with the best policies reported in the literature, i.e., DU and BDU (Thiery and Scherrer, 2009b) in both small and large boards in Table 1. The DT-10 and DT-20 policies, whose weights and features are given in Table 2, are policies learned by CBMPI with D-T features

in the small and large boards, respectively.[12] As shown in Table 1, DT-10 removes $5,000$ lines and outperforms DU, BDU, and DT-20 in the small board. Note that DT-10 is the only policy among these four that has been learned in the small board. In the large board, DT-20 obtains the score of $51,000,000$ and not only outperforms the other three policies, but also achieves the best reported result in the literature (to the best of our knowledge). We observed in our experiments that the learning process in CBMPI has more variance in its performance than the one of CE. We believe this is why in the large board, although the policies learned by CE have similar performance to CBMPI (see Figure 10 (left)), the best policy learned by CBMPI outperforms BDU, the best one learned by CE (see Table 1).

Boards \ Policies	**DU**	**BDU**	**DT-10**	**DT-20**
Small (10×10) board	3800	4200	5000	4300
Large (10×20) board	$31,000,000$	$36,000,000$	$29,000,000$	$51,000,000$

Table 1: Average (over $10,000$ games) score of DU, BDU, DT-10, and DT-20 policies.

feature	**weight**		**feature**	**weight**		**feature**	**weight**	
landing height	-2.18	-2.68	column transitions	-3.31	-6.32	hole depth	-0.81	-0.43
eroded piece cells	2.42	1.38	holes	0.95	2.03	rows w/ holes	-9.65	-9.48
row transitions	-2.17	-2.41	board wells	-2.22	-2.71	diversity	1.27	0.89

Table 2: The weights of the 9 D-T features in the DT-10 (left) and DT-20 (right) policies.

7. Conclusion

In this paper, we considered a dynamic programming (DP) scheme for Markov decision processes, known as modified policy iteration (MPI). We proposed three original approximate MPI (AMPI) algorithms that are extensions of the existing approximate DP (ADP) algorithms: fitted-value iteration, fitted-Q iteration, and classification-based policy iteration. We reported a general error propagation analysis for AMPI that unifies those for approximate policy and value iteration. We instantiated this analysis for the three algorithms that we introduced, which led to a finite-sample analysis of their guaranteed performance. For the last introduced algorithm, CBMPI, our analysis indicated that the main parameter of MPI controls the balance of errors (between value function approximation and estimation of the greedy policy). The role of this parameter was illustrated for all the algorithms on two benchmark problems: Mountain Car and Tetris. Remarkably, in the game of Tetris, CBMPI showed advantages over all previous approaches: it significantly outperforms previous ADP approaches, and is competitive with black-box optimization techniques—the current state of the art for this domain—while using fewer samples. In particular, CBMPI led to what is to our knowledge the currently best Tetris controller, removing $51,000,000$ lines on average. Interesting future work includes **1)** the adaptation and precise analysis of our three algorithms to the computation of non-stationary policies—we recently showed that considering

12. Note that in the standard code by Thiery and Scherrer (2010b), there exist two versions of the feature "board wells" numbered 6 and -6. In our experiments, we used the feature -6 as it is the more computationally efficient of the two.

a variation of AMPI for computing non-stationary policies allows improving the $\frac{1}{(1-\gamma)^2}$ constant (Lesner and Scherrer, 2013)—and **2)** considering problems with large action spaces, for which the methods we have proposed here are likely to have limitation.

Acknowledgments

The experiments were conducted using Grid5000 (https://www.grid5000.fr).

Appendix A. Proof of Lemma 2

Before we start, we recall the following definitions:

$$\begin{aligned}
b_k &= v_k - T_{\pi_{k+1}} v_k, \\
d_k &= v_* - (T_{\pi_k})^m v_{k-1} = v_* - (v_k - \epsilon_k), \\
s_k &= (T_{\pi_k})^m v_{k-1} - v_{\pi_k} = (v_k - \epsilon_k) - v_{\pi_k}.
\end{aligned}$$

Bounding b_k

$$\begin{aligned}
b_k &= v_k - T_{\pi_{k+1}} v_k \\
&= v_k - T_{\pi_k} v_k + T_{\pi_k} v_k - T_{\pi_{k+1}} v_k \\
&\overset{(a)}{\leq} v_k - T_{\pi_k} v_k + \epsilon'_{k+1} \\
&= v_k - \epsilon_k - T_{\pi_k} v_k + \gamma P_{\pi_k} \epsilon_k + \epsilon_k - \gamma P_{\pi_k} \epsilon_k + \epsilon'_{k+1} \\
&\overset{(b)}{=} v_k - \epsilon_k - T_{\pi_k}(v_k - \epsilon_k) + (I - \gamma P_{\pi_k})\epsilon_k + \epsilon'_{k+1}.
\end{aligned} \tag{25}$$

Using the definition of x_k, i.e.,

$$x_k \overset{\Delta}{=} (I - \gamma P_{\pi_k})\epsilon_k + \epsilon'_{k+1}, \tag{26}$$

we may write Equation 25 as

$$\begin{aligned}
b_k &\leq v_k - \epsilon_k - T_{\pi_k}(v_k - \epsilon_k) + x_k \\
&\overset{(c)}{=} (T_{\pi_k})^m v_{k-1} - T_{\pi_k}(T_{\pi_k})^m v_{k-1} + x_k \\
&= (T_{\pi_k})^m v_{k-1} - (T_{\pi_k})^m (T_{\pi_k} v_{k-1}) + x_k \\
&\overset{(d)}{=} (\gamma P_{\pi_k})^m (v_{k-1} - T_{\pi_k} v_{k-1}) + x_k \\
&= (\gamma P_{\pi_k})^m b_{k-1} + x_k.
\end{aligned} \tag{27}$$

(a) From the definition of ϵ'_{k+1}, we have $\forall \pi' \;\; T_{\pi'} v_k \leq T_{\pi_{k+1}} v_k + \epsilon'_{k+1}$, thus this inequality holds also for $\pi' = \pi_k$.
(b) This step is due to the fact that for every v and v', we have $T_{\pi_k}(v + v') = T_{\pi_k} v + \gamma P_{\pi_k} v'$.
(c) This is from the definition of ϵ_k, i.e., $v_k = (T_{\pi_k})^m v_{k-1} + \epsilon_k$.
(d) This step is due to the fact that for every v and v', any m, we have $(T_{\pi_k})^m v - (T_{\pi_k})^m v' = (\gamma P_{\pi_k})^m (v - v')$.

Bounding d_k Define

$$g_{k+1} \stackrel{\Delta}{=} T_{\pi_{k+1}} v_k - (T_{\pi_{k+1}})^m v_k. \tag{28}$$

Then,

$$\begin{aligned}
d_{k+1} &= v_* - (T_{\pi_{k+1}})^m v_k \\
&= T_{\pi_*} v_* - T_{\pi_*} v_k + T_{\pi_*} v_k - T_{\pi_{k+1}} v_k + T_{\pi_{k+1}} v_k - (T_{\pi_{k+1}})^m v_k \\
&\stackrel{(a)}{\leq} \gamma P_{\pi_*}(v_* - v_k) + \epsilon'_{k+1} + g_{k+1} \\
&= \gamma P_{\pi_*}(v_* - v_k) + \gamma P_{\pi_*} \epsilon_k - \gamma P_{\pi_*} \epsilon_k + \epsilon'_{k+1} + g_{k+1} \\
&\stackrel{(b)}{=} \gamma P_{\pi_*}\big(v_* - (v_k - \epsilon_k)\big) + y_k + g_{k+1} \\
&= \gamma P_{\pi_*} d_k + y_k + g_{k+1} \\
&\stackrel{(c)}{=} \gamma P_{\pi_*} d_k + y_k + \sum_{j=1}^{m-1} (\gamma P_{\pi_{k+1}})^j b_k.
\end{aligned} \tag{29}$$

(a) This step is from the definition of ϵ'_{k+1} (see step **(a)** in bounding b_k) and that of g_{k+1} in Equation 28.
(b) This is from the definition of y_k, i.e.,

$$y_k \stackrel{\Delta}{=} -\gamma P_{\pi_*} \epsilon_k + \epsilon'_{k+1}. \tag{30}$$

(c) This step comes from rewriting g_{k+1} as

$$\begin{aligned}
g_{k+1} &= T_{\pi_{k+1}} v_k - (T_{\pi_{k+1}})^m v_k \\
&= \sum_{j=1}^{m-1} \left[(T_{\pi_{k+1}})^j v_k - (T_{\pi_{k+1}})^{j+1} v_k \right] \\
&= \sum_{j=1}^{m-1} \left[(T_{\pi_{k+1}})^j v_k - (T_{\pi_{k+1}})^j (T_{\pi_{k+1}} v_k) \right] \\
&= \sum_{j=1}^{m-1} (\gamma P_{\pi_{k+1}})^j (v_k - T_{\pi_{k+1}} v_k) \\
&= \sum_{j=1}^{m-1} (\gamma P_{\pi_{k+1}})^j b_k.
\end{aligned} \tag{31}$$

Bounding s_k With some slight abuse of notation, we have

$$v_{\pi_k} = (T_{\pi_k})^\infty v_k$$

and thus:

$$s_k = (T_{\pi_k})^m v_{k-1} - v_{\pi_k}$$

$$
\begin{aligned}
&\overset{(a)}{=} (T_{\pi_k})^m v_{k-1} - (T_{\pi_k})^\infty v_{k-1} \\
&= (T_{\pi_k})^m v_{k-1} - (T_{\pi_k})^m (T_{\pi_k})^\infty v_{k-1} \\
&= (\gamma P_{\pi_k})^m \big(v_{k-1} - (T_{\pi_k})^\infty v_{k-1}\big) \\
&= (\gamma P_{\pi_k})^m \sum_{j=0}^{\infty} \big[(T_{\pi_k})^j v_{k-1} - (T_{\pi_k})^{j+1} v_{k-1}\big] \\
&= (\gamma P_{\pi_k})^m \sum_{j=0}^{\infty} \big[(T_{\pi_k})^j v_{k-1} - (T_{\pi_k})^j T_{\pi_k} v_{k-1}\big] \\
&= (\gamma P_{\pi_k})^m \Big(\sum_{j=0}^{\infty} (\gamma P_{\pi_k})^j\Big)(v_{k-1} - T_{\pi_k} v_{k-1}) \\
&= (\gamma P_{\pi_k})^m (I - \gamma P_{\pi_k})^{-1} (v_{k-1} - T_{\pi_k} v_{k-1}) \\
&= (\gamma P_{\pi_k})^m (I - \gamma P_{\pi_k})^{-1} b_{k-1}. \qquad (32)
\end{aligned}
$$

(a) For any v, we have $v_{\pi_k} = (T_{\pi_k})^\infty v$. This step follows by setting $v = v_{k-1}$, i.e., $v_{\pi_k} = (T_{\pi_k})^\infty v_{k-1}$.

Appendix B. Proof of Lemma 4

We begin by focusing our analysis on AMPI. Here we are interested in bounding the loss $l_k = v_* - v_{\pi_k} = d_k + s_k$.

By induction, from Equations 27 and 29, we obtain

$$b_k \le \sum_{i=1}^{k} \Gamma^{m(k-i)} x_i + \Gamma^{mk} b_0, \qquad (33)$$

$$d_k \le \sum_{j=0}^{k-1} \Gamma^{k-1-j} \Big(y_j + \sum_{l=1}^{m-1} \Gamma^l b_j\Big) + \Gamma^k d_0. \qquad (34)$$

in which we have used the notation introduced in Definition 3. In Equation 34, we also used the fact that from Equation 31, we may write $g_{k+1} = \sum_{j=1}^{m-1} \Gamma^j b_k$. Moreover, we may rewrite Equation 32 as

$$s_k = \Gamma^m \sum_{j=0}^{\infty} \Gamma^j b_{k-1} = \sum_{j=0}^{\infty} \Gamma^{m+j} b_{k-1}. \qquad (35)$$

Bounding l_k From Equations 33 and 34, we may write

$$
\begin{aligned}
d_k &\le \sum_{j=0}^{k-1} \Gamma^{k-1-j} \left(y_j + \sum_{l=1}^{m-1} \Gamma^l \Big(\sum_{i=1}^{j} \Gamma^{m(j-i)} x_i + \Gamma^{mj} b_0\Big)\right) + \Gamma^k d_0 \\
&= \sum_{i=1}^{k} \Gamma^{i-1} y_{k-i} + \sum_{j=0}^{k-1} \sum_{l=1}^{m-1} \sum_{i=1}^{j} \Gamma^{k-1-j+l+m(j-i)} x_i + z_k, \qquad (36)
\end{aligned}
$$

where we used the following definition

$$z_k \stackrel{\Delta}{=} \sum_{j=0}^{k-1} \sum_{l=1}^{m-1} \Gamma^{k-1+l+j(m-1)} b_0 + \Gamma^k d_0 = \sum_{i=k}^{mk-1} \Gamma^i b_0 + \Gamma^k d_0.$$

The triple sum involved in Equation 36 may be written as

$$\begin{aligned}
\sum_{j=0}^{k-1} \sum_{l=1}^{m-1} \sum_{i=1}^{j} \Gamma^{k-1-j+l+m(j-i)} x_i &= \sum_{i=1}^{k-1} \sum_{j=i}^{k-1} \sum_{l=1}^{m-1} \Gamma^{k-1+l+j(m-1)-mi} x_i \\
&= \sum_{i=1}^{k-1} \sum_{j=mi+k-i}^{mk-1} \Gamma^{j-mi} x_i \\
&= \sum_{i=1}^{k-1} \sum_{j=k-i}^{m(k-i)-1} \Gamma^{j} x_i \\
&= \sum_{i=1}^{k-1} \sum_{j=i}^{mi-1} \Gamma^{j} x_{k-i}. \qquad (37)
\end{aligned}$$

Using Equation 37, we may write Equation 36 as

$$d_k \leq \sum_{i=1}^{k} \Gamma^{i-1} y_{k-i} + \sum_{i=1}^{k-1} \sum_{j=i}^{mi-1} \Gamma^j x_{k-i} + z_k. \qquad (38)$$

Similarly, from Equations 35 and 33, we have

$$\begin{aligned}
s_k &\leq \sum_{j=0}^{\infty} \Gamma^{m+j} \Big(\sum_{i=1}^{k-1} \Gamma^{m(k-1-i)} x_i + \Gamma^{m(k-1)} b_0 \Big) \\
&= \sum_{j=0}^{\infty} \Big(\sum_{i=1}^{k-1} \Gamma^{m+j+m(k-1-i)} x_i + \Gamma^{m+j+m(k-1)} b_0 \Big) \\
&= \sum_{i=1}^{k-1} \sum_{j=0}^{\infty} \Gamma^{j+m(k-i)} x_i + \sum_{j=0}^{\infty} \Gamma^{j+mk} b_0 = \sum_{i=1}^{k-1} \sum_{j=0}^{\infty} \Gamma^{j+mi} x_{k-i} + \sum_{j=mk}^{\infty} \Gamma^j b_0 \\
&= \sum_{i=1}^{k-1} \sum_{j=mi}^{\infty} \Gamma^j x_{k-i} + z'_k, \qquad (39)
\end{aligned}$$

where we used the following definition

$$z'_k \stackrel{\Delta}{=} \sum_{j=mk}^{\infty} \Gamma^j b_0.$$

Finally, using the bounds in Equations 38 and 39, we obtain the following bound on the loss

$$l_k \leq d_k + s_k$$

$$\leq \sum_{i=1}^{k} \Gamma^{i-1} y_{k-i} + \sum_{i=1}^{k-1} \Big(\sum_{j=i}^{mi-1} \Gamma^j + \sum_{j=mi}^{\infty} \Gamma^j \Big) x_{k-i} + z_k + z'_k$$
$$= \sum_{i=1}^{k} \Gamma^{i-1} y_{k-i} + \sum_{i=1}^{k-1} \sum_{j=i}^{\infty} \Gamma^j x_{k-i} + \eta_k, \tag{40}$$

where we used the following definition

$$\eta_k \stackrel{\Delta}{=} z_k + z'_k = \sum_{j=k}^{\infty} \Gamma^j b_0 + \Gamma^k d_0. \tag{41}$$

Note that we have the following relation between b_0 and d_0

$$\begin{aligned} b_0 &= v_0 - T_{\pi_1} v_0 \\ &= v_0 - v_* + T_{\pi_*} v_* - T_{\pi_*} v_0 + T_{\pi_*} v_0 - T_{\pi_1} v_0 \\ &\leq (I - \gamma P_{\pi_*})(-d_0) + \epsilon'_1, \end{aligned} \tag{42}$$

In Equation 42, we used the fact that $v_* = T_{\pi_*} v_*$, $\epsilon_0 = 0$, and $T_{\pi_*} v_0 - T_{\pi_1} v_0 \leq \epsilon'_1$ (this is because the policy π_1 is ϵ'_1-greedy w.r.t. v_0). As a result, we may write $|\eta_k|$ either as

$$\begin{aligned} |\eta_k| &\leq \sum_{j=k}^{\infty} \Gamma^j \big[(I - \gamma P_{\pi_*}) |d_0| + |\epsilon'_1| \big] + \Gamma^k |d_0| \\ &\leq \sum_{j=k}^{\infty} \Gamma^j \big[(I + \Gamma^1) |d_0| + |\epsilon'_1| \big] + \Gamma^k |d_0| \\ &= 2 \sum_{j=k}^{\infty} \Gamma^j |d_0| + \sum_{j=k}^{\infty} \Gamma^j |\epsilon'_1|, \end{aligned} \tag{43}$$

or using the fact that from Equation 42, we have $d_0 \leq (I - \gamma P_{\pi_*})^{-1}(-b_0 + \epsilon'_1)$, as

$$\begin{aligned} |\eta_k| &\leq \sum_{j=k}^{\infty} \Gamma^j |b_0| + \Gamma^k \sum_{j=0}^{\infty} (\gamma P_{\pi_*})^j \big(|b_0| + |\epsilon'_1| \big) \\ &= \sum_{j=k}^{\infty} \Gamma^j |b_0| + \Gamma^k \sum_{j=0}^{\infty} \Gamma^j \big(|b_0| + |\epsilon'_1| \big) \\ &= 2 \sum_{j=k}^{\infty} \Gamma^j |b_0| + \sum_{j=k}^{\infty} \Gamma^j |\epsilon'_1|. \end{aligned} \tag{44}$$

Now, using the definitions of x_k and y_k in Equations 26 and 30, the bound on $|\eta_k|$ in Equation 43 or 44, and the fact that $\epsilon_0 = 0$, we obtain

$$|l_k| \leq \sum_{i=1}^{k} \Gamma^{i-1} \big[\Gamma^1 |\epsilon_{k-i}| + |\epsilon'_{k-i+1}| \big] + \sum_{i=1}^{k-1} \sum_{j=i}^{\infty} \Gamma^j \big[(I + \Gamma^1) |\epsilon_{k-i}| + |\epsilon'_{k-i+1}| \big] + |\eta_k|$$

$$= \sum_{i=1}^{k-1} \left(\Gamma^i + \sum_{j=i}^{\infty} (\Gamma^j + \Gamma^{j+1}) \right) |\epsilon_{k-i}| + \Gamma^k |\epsilon_0| \tag{45}$$

$$+ \sum_{i=1}^{k-1} \left(\Gamma^{i-1} + \sum_{j=i}^{\infty} \Gamma^j \right) |\epsilon'_{k-i+1}| + \Gamma^{k-1} |\epsilon'_1| + \sum_{j=k}^{\infty} \Gamma^j |\epsilon'_1| + h(k)$$

$$= 2 \sum_{i=1}^{k-1} \sum_{j=i}^{\infty} \Gamma^j |\epsilon_{k-i}| + \sum_{i=1}^{k-1} \sum_{j=i-1}^{\infty} \Gamma^j |\epsilon'_{k-i+1}| + \sum_{j=k-1}^{\infty} \Gamma^j |\epsilon'_1| + h(k)$$

$$= 2 \sum_{i=1}^{k-1} \sum_{j=i}^{\infty} \Gamma^j |\epsilon_{k-i}| + \sum_{i=0}^{k-1} \sum_{j=i}^{\infty} \Gamma^j |\epsilon'_{k-i}| + h(k), \tag{46}$$

where we used the following definition

$$h(k) \triangleq 2 \sum_{j=k}^{\infty} \Gamma^j |d_0| \quad \text{or} \quad h(k) \triangleq 2 \sum_{j=k}^{\infty} \Gamma^j |b_0|,$$

depending on whether one uses Equation 43 or Equation 44.

We end this proof by adapting the error propagation to CBMPI. As expressed by Equations 20 and 21 in Section 4, an analysis of CBMPI can be deduced from that we have just done by replacing v_k with the auxiliary variable $w_k = (T_{\pi_k})^m v_{k-1}$ and ϵ_k with $(\gamma P_{\pi_k})^m \epsilon_{k-1} = \Gamma^m \epsilon_{k-1}$. Therefore, using the fact that $\epsilon_0 = 0$, we can rewrite the bound of Equation 46 for CBMPI as follows:

$$l_k \leq 2 \sum_{i=1}^{k-1} \sum_{j=i}^{\infty} \Gamma^{j+m} |\epsilon_{k-i-1}| + \sum_{i=0}^{k-1} \sum_{j=i}^{\infty} \Gamma^j |\epsilon'_{k-i}| + h(k)$$

$$= 2 \sum_{i=1}^{k-2} \sum_{j=m+i}^{\infty} \Gamma^j |\epsilon_{k-i-1}| + \sum_{i=0}^{k-1} \sum_{j=i}^{\infty} \Gamma^j |\epsilon'_{k-i}| + h(k). \tag{47}$$

Appendix C. Proof of Lemma 6

For any integer t and vector z, the definition of Γ^t and Hölder's inequality imply that

$$\rho \Gamma^t |z| = \left\| \Gamma^t |z| \right\|_{1,\rho} \leq \gamma^t c_q(t) \|z\|_{q',\mu} = \gamma^t c_q(t) \left(\mu |z|^{q'} \right)^{\frac{1}{q'}}. \tag{48}$$

We define

$$K \triangleq \sum_{l=1}^{n} \xi_l \left(\sum_{i \in \mathcal{I}_l} \sum_{j \in \mathcal{J}_i} \gamma^j \right),$$

where $\{\xi_l\}_{l=1}^n$ is a set of non-negative numbers that we will specify later. We now have

$$\|f\|_{p,\rho}^p = \rho |f|^p$$

$$\leq K^p \rho \left(\frac{\sum_{l=1}^n \sum_{i \in \mathcal{I}_l} \sum_{j \in \mathcal{J}_i} \Gamma^j |g_i|}{K} \right)^p = K^p \rho \left(\frac{\sum_{l=1}^n \xi_l \sum_{i \in \mathcal{I}_l} \sum_{j \in \mathcal{J}_i} \Gamma^j \left(\frac{|g_i|}{\xi_l} \right)}{K} \right)^p$$

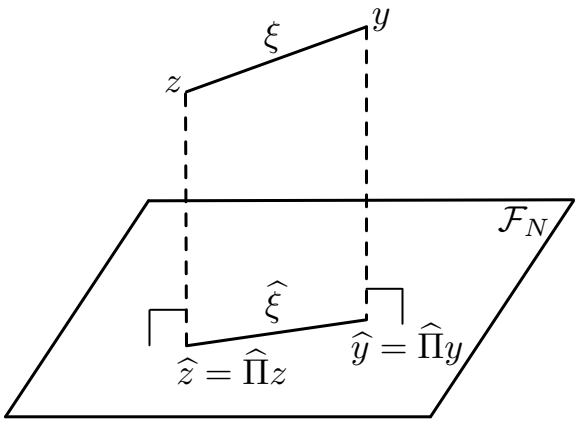


Figure 11: The notations used in the proof.

$$
\begin{aligned}
&\overset{(a)}{\leq} K^p \rho \frac{\sum_{l=1}^n \xi_l \sum_{i\in\mathcal{I}_l} \sum_{j\in\mathcal{J}_i} \Gamma^j \left(\frac{|g_i|}{\xi_l}\right)^p}{K} = K^p \frac{\sum_{l=1}^n \xi_l \sum_{i\in\mathcal{I}_l} \sum_{j\in\mathcal{J}_i} \rho\Gamma^j \left(\frac{|g_i|}{\xi_l}\right)^p}{K} \\
&\overset{(b)}{\leq} K^p \frac{\sum_{l=1}^n \xi_l \sum_{i\in\mathcal{I}_l} \sum_{j\in\mathcal{J}_i} \gamma^j c_q(j) \left(\mu\left(\frac{|g_i|}{\xi_l}\right)^{pq'}\right)^{\frac{1}{q'}}}{K} \\
&= K^p \frac{\sum_{l=1}^n \xi_l \sum_{i\in\mathcal{I}_l} \sum_{j\in\mathcal{J}_i} \gamma^j c_q(j) \left(\frac{\|g_i\|_{pq',\mu}}{\xi_l}\right)^p}{K} \\
&\leq K^p \frac{\sum_{l=1}^n \xi_l \left(\sum_{i\in\mathcal{I}_l} \sum_{j\in\mathcal{J}_i} \gamma^j c_q(j)\right) \left(\frac{\sup_{i\in\mathcal{I}_l} \|g_i\|_{pq',\mu}}{\xi_l}\right)^p}{K} \\
&\overset{(c)}{=} K^p \frac{\sum_{l=1}^n \xi_l \left(\sum_{i\in\mathcal{I}_l} \sum_{j\in\mathcal{J}_i} \gamma^j\right) \mathcal{C}_q(l) \left(\frac{\sup_{i\in\mathcal{I}_l} \|g_i\|_{pq',\mu}}{\xi_l}\right)^p}{K},
\end{aligned}
$$

where **(a)** results from Jensen's inequality, **(b)** from Equation 48, and **(c)** from the definition of $\mathcal{C}_q(l)$. Now, by setting $\xi_l = \left(\mathcal{C}_q(l)\right)^{1/p} \sup_{i\in\mathcal{I}_l} \|g_i\|_{pq',\mu}$, we obtain

$$
\|f\|_{p,\rho}^p \leq K^p \frac{\sum_{l=1}^n \xi_l \left(\sum_{i\in\mathcal{I}_l} \sum_{j\in\mathcal{J}_i} \gamma^j\right)}{K} = K^p,
$$

where the last step follows from the definition of K.

Appendix D. Proof of Lemma 11

Let $\widehat{\mu}$ be the empirical distribution corresponding to states $s^{(1)}, \ldots, s^{(n)}$. Let us define two N-dimensional vectors $z = \left(\left[(T_{\pi_k})^m v_{k-1}\right](s^{(1)}), \ldots, \left[(T_{\pi_k})^m v_{k-1}\right](s^{(N)})\right)^\top$ and $y = \left(\widehat{v}_k(s^{(1)}), \ldots, \widehat{v}_k(s^{(N)})\right)^\top$ and their orthogonal projections onto the vector space $\mathcal{F}_N$ as $\widehat{z} = \widehat{\Pi} z$ and $\widehat{y} = \widehat{\Pi} y = \left(\widetilde{v}_k(s^{(1)}), \ldots, \widetilde{v}_k(s^{(N)})\right)^\top$, where $\widetilde{v}_k$ is the result of linear regression and its truncation (by $V_{\max}$) is v_k, i.e., $v_k = \mathbb{T}(\widetilde{v}_k)$ (see Figure 11). What we are interested in is

to find a bound on the regression error $\|z - \widehat{y}\|$ (the difference between the target function z and the result of the regression $\widehat{y}$). We may decompose this error as

$$\|z - \widehat{y}\|_{2,\widehat{\mu}} \leq \|\widehat{z} - \widehat{y}\|_{2,\widehat{\mu}} + \|z - \widehat{z}\|_{2,\widehat{\mu}} = \|\widehat{\xi}\|_{2,\widehat{\mu}} + \|z - \widehat{z}\|_{2,\widehat{\mu}}, \tag{49}$$

where $\widehat{\xi} = \widehat{z} - \widehat{y}$ is the projected noise (estimation error) $\widehat{\xi} = \widehat{\Pi}\xi$, with the noise vector $\xi = z - y$ defined as $\xi_i = \big[(T_{\pi_k})^m v_{k-1}\big](s^{(i)}) - \widehat{v}_k(s^{(i)})$. It is easy to see that noise is zero mean, i.e., $\mathbb{E}[\xi_i] = 0$ and is bounded by $2V_{\max}$, i.e., $|\xi_i| \leq 2V_{\max}$. We may write the estimation error as

$$\|\widehat{z} - \widehat{y}\|^2_{2,\widehat{\mu}} = \|\widehat{\xi}\|^2_{2,\widehat{\mu}} = \langle \widehat{\xi}, \widehat{\xi} \rangle = \langle \xi, \widehat{\xi} \rangle,$$

where the last equality follows from the fact that $\widehat{\xi}$ is the orthogonal projection of ξ. Since $\widehat{\xi} \in \mathcal{F}_N$, let $f_\alpha \in \mathcal{F}$ be any function in the function space $\mathcal{F}$,[13] whose values at $\{s^{(i)}\}_{i=1}^N$ equals to $\{\widehat{\xi}_i\}_{i=1}^N$. By application of a variation of Pollard's inequality (Györfi et al., 2002), we obtain

$$\langle \xi, \widehat{\xi} \rangle = \frac{1}{N} \sum_{i=1}^N \xi_i f_\alpha(s^{(i)}) \leq 4V_{\max} \|\widehat{\xi}\|_{2,\widehat{\mu}} \sqrt{\frac{2}{N} \log\left(\frac{3(9e^2N)^{d+1}}{\delta'}\right)},$$

with probability at least $1 - \delta'$. Thus, we have

$$\|\widehat{z} - \widehat{y}\|_{2,\widehat{\mu}} = \|\widehat{\xi}\|_{2,\widehat{\mu}} \leq 4V_{\max} \sqrt{\frac{2}{N} \log\left(\frac{3(9e^2N)^{d+1}}{\delta'}\right)}. \tag{50}$$

From Equations 49 and 50, we have

$$\|(T_{\pi_k})^m v_{k-1} - \widetilde{v}_k\|_{2,\widehat{\mu}} \leq \|(T_{\pi_k})^m v_{k-1} - \widehat{\Pi}(T_{\pi_k})^m v_{k-1}\|_{2,\widehat{\mu}} + 4V_{\max} \sqrt{\frac{2}{N} \log\left(\frac{3(9e^2N)^{d+1}}{\delta'}\right)}. \tag{51}$$

Now in order to obtain a random design bound, we first define $f_{\widehat{\alpha}_*} \in \mathcal{F}$ as $f_{\widehat{\alpha}_*}(s^{(i)}) = \big[\widehat{\Pi}(T_{\pi_k})^m v_{k-1}\big](s^{(i)})$, and then define $f_{\alpha_*} = \Pi(T_{\pi_k})^m v_{k-1}$ that is the best approximation (w.r.t. μ) of the target function $(T_{\pi_k})^m v_{k-1}$ in $\mathcal{F}$. Since $f_{\widehat{\alpha}_*}$ is the minimizer of the empirical loss, any function in $\mathcal{F}$ different than $f_{\widehat{\alpha}_*}$ has a bigger empirical loss, thus we have

$$\begin{aligned} \|f_{\widehat{\alpha}_*} - (T_{\pi_k})^m v_{k-1}\|_{2,\widehat{\mu}} &\leq \|f_{\alpha_*} - (T_{\pi_k})^m v_{k-1}\|_{2,\widehat{\mu}} \\ &\leq 2\|f_{\alpha_*} - (T_{\pi_k})^m v_{k-1}\|_{2,\mu} \\ &\quad + 12\Big(V_{\max} + \|\alpha_*\|_2 \sup_x \|\phi(x)\|_2\Big)\sqrt{\frac{2}{N} \log \frac{3}{\delta'}} \end{aligned} \tag{52}$$

with probability at least $1-\delta'$, where the second inequality is the application of a variation of Theorem 11.2 in Györfi *et al.* (2002) with $\|f_{\alpha_*} - (T_{\pi_k})^m v_{k-1}\|_\infty \leq V_{\max} + \|\alpha^*\|_2 \sup_x \|\phi(x)\|_2$. Similarly, we can write the left-hand-side of Equation 51 as

13. We should discriminate between the linear function space $\mathcal{F} = \big\{f_\alpha \mid \alpha \in \mathbb{R}^d \text{ and } f_\alpha(\cdot) = \phi(\cdot)^\top \alpha\big\}$, where $\phi(\cdot) = \big(\varphi_1(\cdot), \ldots, \varphi_d(\cdot)\big)^\top$, and its corresponding linear vector space $\mathcal{F}_N = \{\Phi\alpha,\ \alpha \in \mathbb{R}^d\} \subset \mathbb{R}^N$, where $\Phi = \big[\phi(s^{(1)})^\top; \ldots; \phi(s^{(N)})^\top\big]$.

$$2\|(T_{\pi_k})^m v_{k-1} - \widetilde{v}_k\|_{2,\widehat{\mu}} \geq 2\|(T_{\pi_k})^m v_{k-1} - \mathbb{T}(\widetilde{v}_k)\|_{2,\widehat{\mu}}$$
$$\geq \|(T_{\pi_k})^m v_{k-1} - \mathbb{T}(\widetilde{v}_k)\|_{2,\mu} - 24V_{\max}\sqrt{\frac{2}{N}\Lambda(N,d,\delta')} \qquad (53)$$

with probability at least $1-\delta'$, where $\Lambda(N,d,\delta') = 2(d+1)\log N + \log\frac{e}{\delta'} + \log\left(9(12e)^{2(d+1)}\right)$. Putting together Equations 51, 52, and 53 and using the fact that $\mathbb{T}(\widetilde{v}_k) = v_k$, we obtain

$$\|\epsilon_k\|_{2,\mu} = \|(T_{\pi_k})^m v_{k-1} - v_k\|_{2,\mu}$$
$$\leq 2\Bigg(2\|(T_{\pi_k})^m v_{k-1} - f_{\alpha_*}\|_{2,\mu}$$
$$+ 12\Big(V_{\max} + \|\alpha_*\|_2 \sup_x \|\phi(x)\|_2\Big)\sqrt{\frac{2}{N}\log\frac{3}{\delta'}} + 4V_{\max}\sqrt{\frac{2}{N}\log\left(\frac{3(9e^2N)^{d+1}}{\delta'}\right)}\Bigg)$$
$$+ 24V_{\max}\sqrt{\frac{2}{N}\Lambda(N,d,\delta')}.$$

The result follows by setting $\delta = 3\delta'$ and some simplifications.

Appendix E. Proof of Lemma 12

Proof We prove the following series of inequalities:

$$\begin{aligned}
||\epsilon'_k||_{1,\mu} &\overset{(a)}{\leq} ||\epsilon'_k||_{1,\widehat{\mu}} + e'_3(N,\delta') && \text{w.p. } 1-\delta' \\
&\overset{(b)}{=} \frac{1}{N}\sum_{i=1}^{N}\left[\max_{a\in\mathcal{A}}\big(T_a v_{k-1}\big)(s^{(i)}) - \big(T_{\pi_k} v_{k-1}\big)(s^{(i)})\right] + e'_3(N,\delta') \\
&\overset{(c)}{\leq} \frac{1}{N}\sum_{i=1}^{N}\left[\max_{a\in\mathcal{A}}\big(T_a v_{k-1}\big)(s^{(i)}) - \frac{1}{M}\sum_{j=1}^{M}\big(\widehat{T}^{(j)}_{\pi_k} v_{k-1}\big)(s^{(i)})\right] + e'_3(N,\delta') + e'_4(N,M,\delta') && \text{w.p. } 1-2\delta' \\
&\overset{(d)}{=} \frac{1}{N}\sum_{i=1}^{N}\left[\max_{a\in\mathcal{A}}\big(T_a v_{k-1}\big)(s^{(i)}) - \max_{a'\in\mathcal{A}}\frac{1}{M}\sum_{j=1}^{M}\big(\widehat{T}^{(j)}_{a'} v_{k-1}\big)(s^{(i)})\right] + e'_3(N,\delta') + e'_4(N,M,\delta') \\
&\overset{(e)}{\leq} \frac{1}{N}\sum_{i=1}^{N}\left\{\max_{a\in\mathcal{A}}\left[\big(T_a v_{k-1}\big)(s^{(i)}) - \frac{1}{M}\sum_{j=1}^{M}\big(\widehat{T}^{(j)}_{a} v_{k-1}\big)(s^{(i)})\right]\right\} + e'_3(N,\delta') + e'_4(N,M,\delta') \\
&\overset{(f)}{\leq} e'_3(N,\delta') + e'_4(N,M,\delta') + e'_5(M,N,\delta') && \text{w.p. } 1-3\delta'
\end{aligned}$$

(a) This step is the result of the following lemma.

Lemma 17 *Let Π be the policy space of the policies obtained by Equation 4 from the truncation (by $V_{\max}$) of the function space $\mathcal{F}$, with finite VC-dimension $h = VC(\Pi) < \infty$. Let $N > 0$ be the number of states in the rollout set $\mathcal{D}_k$, drawn i.i.d. from the state distribution μ. Then, we have*

$$\mathbb{P}_{\mathcal{D}_k}\left[\sup_{\pi\in\Pi}\Big|||\epsilon'_k(\pi)||_{1,\widehat{\mu}} - ||\epsilon'_k(\pi)||_{1,\mu}\Big| > e'_3(N,\delta')\right] \leq \delta' \ ,$$

with $e'_3(N,\delta') = 16V_{\max}\sqrt{\frac{2}{N}(h\log\frac{eN}{h} + \log\frac{8}{\delta'})}$.

Proof The proof is similar to the proof of Lemma 1 in Lazaric *et al.* (2010a). ■

(b) This is from the definition of $||\epsilon'_k||_{1,\widehat{\mu}}$.

(c) This step is the result of bounding

$$\sup_{\pi\in\Pi}\left[\frac{1}{MN}\sum_{i=1}^{N}\sum_{j=1}^{M}\big(\widehat{T}_{\pi}^{(j)}v_{k-1}\big)(s^{(i)}) - \frac{1}{MN}\sum_{i=1}^{N}\sum_{j=1}^{M}\big(T_{\pi}v_{k-1}\big)(s^{(i)})\right]$$

by $e'_4(N,M,\delta')$. The supremum over all the policies in the policy space Π is due to the fact that π_k is a random object whose randomness comes from all the randomly generated samples at the k'th iteration (i.e., the states in the rollout set and all the generated rollouts). We bound this term using the following lemma.

Lemma 18 *Let Π be the policy space of the policies obtained by Equation 4 from the truncation (by $V_{\max}$) of the function space $\mathcal{F}$, with finite VC-dimension $h = VC(\Pi) < \infty$. Let $\{s^{(i)}\}_{i=1}^{N}$ be N states sampled i.i.d. from the distribution μ. For each sampled state $s^{(i)}$, we take the action suggested by policy π, M times, and observe the next states $\{s^{(i,j)}\}_{j=1}^{M}$. Then, we have*

$$\mathbb{P}\left[\sup_{\pi\in\Pi}\left|\frac{1}{N}\sum_{i=1}^{N}\frac{1}{M}\sum_{j=1}^{M}\big[r\big(s^{(i)},\pi(s^{(i)}\big)+\gamma v_{k-1}\big(s^{(i,j)}\big)\big] - \frac{1}{N}\sum_{i=1}^{N}\big(T_{\pi}v_{k-1}\big)(s^{(i)})\right| > e'_4(N,M,\delta')\right] \le \delta' ,$$

with $e'_4(N,M,\delta') = 8V_{\max}\sqrt{\frac{2}{MN}\big(h\log\frac{eMN}{h}+\log\frac{8}{\delta'}\big)}$.

Proof The proof is similar to the proof of Lemma 4 in Lazaric *et al.* (2010b). ■

(d) This step is from the definition of π_k in the AMPI-V algorithm (Equation 4).

(e) This step is algebra, replacing two maximums with one.

(f) This step follows from applying Chernoff-Hoeffding to bound

$$\big(T_{a_*^{(i)}}v_{k-1}\big)(s^{(i)}) - \frac{1}{M}\sum_{j=1}^{M}\big(\widehat{T}_{a_*^{(i)}}^{(j)}v_{k-1}\big)(s^{(i)}),$$

for each $i = 1,\ldots,N$, by $e'_5(M,\delta'') = V_{\max}\sqrt{\frac{2\log(1/\delta'')}{M}}$, followed by a union bound, which gives us $e'_5(M,N,\delta') = V_{\max}\sqrt{\frac{2\log(N/\delta')}{M}}$. Note that the fixed action $a_*^{(i)}$ is defined as

$$a_*^{(i)} = \operatorname*{argmax}_{a\in\mathcal{A}}\Big[\big(T_a v_{k-1}\big)(s^{(i)}) - \frac{1}{M}\sum_{j=1}^{M}\big(\widehat{T}_a^{(j)}v_{k-1}\big)(s^{(i)})\Big].$$

The final statement of the theorem follows by setting $\delta = 3\delta'$. ■

Appendix F. Proof of Lemma 13

The proof of this lemma is similar to the proof of Theorem 1 in Lazaric *et al.* (2010a). Before stating the proof, we report the following two lemmas that are used in the proof.

Lemma 19 *Let Π be a policy space with finite VC-dimension $h = VC(\Pi) < \infty$ and N' be the number of states in the rollout set $\mathcal{D}'_{k-1}$ drawn i.i.d. from the state distribution μ. Then we have*

$$\mathbb{P}_{\mathcal{D}'_{k-1}}\left[\sup_{\pi\in\Pi}\left|\mathcal{L}^{\Pi}_{k-1}(\widehat{\mu};\pi) - \mathcal{L}^{\Pi}_{k-1}(\mu;\pi)\right| > \epsilon\right] \leq \delta \ ,$$

with $\epsilon = 16Q_{\max}\sqrt{\frac{2}{N'}\left(h\log\frac{eN'}{h} + \log\frac{8}{\delta}\right)}$.

Proof This is a restatement of Lemma 1 in Lazaric *et al.* (2010a). ■

Lemma 20 *Let Π be a policy space with finite VC-dimension $h = VC(\Pi) < \infty$ and $s^{(1)}, \ldots, s^{(N')}$ be an arbitrary sequence of states. Assume that at each state, we simulate M independent rollouts. We have*

$$\mathbb{P}\left[\sup_{\pi\in\Pi}\left|\frac{1}{N'}\sum_{i=1}^{N'}\frac{1}{M}\sum_{j=1}^{M}R^{j}_{k-1}\big(s^{(i,j)},\pi(s^{(i,j)})\big) - \frac{1}{N'}\sum_{i=1}^{N'}Q_{k-1}\big(s^{(i,j)},\pi(s^{(i,j)})\big)\right| > \epsilon\right] \leq \delta \ ,$$

with $\epsilon = 8Q_{\max}\sqrt{\frac{2}{MN'}\left(h\log\frac{eMN'}{h} + \log\frac{8}{\delta}\right)}$.

Proof The proof is similar to the one for Lemma 19. ■

Proof (Lemma 13) Let $a^*(\cdot) \in \operatorname{argmax}_{a\in\mathcal{A}} Q_{k-1}(\cdot, a)$ be a greedy action. To simplify the notation, we remove the dependency of a^* on states and use a^* instead of $a^*(s^{(i)})$ in the following. We prove the following series of inequalities:

$$\begin{aligned}
\mathcal{L}^{\Pi}_{k-1}(\mu;\pi_k) &\overset{(a)}{\leq} \mathcal{L}^{\Pi}_{k-1}(\widehat{\mu};\pi_k) + e'_1(N',\delta) && \text{w.p. } 1-\delta' \\
&= \frac{1}{N'}\sum_{i=1}^{N'}\Big[Q_{k-1}(s^{(i)},a^*) - Q_{k-1}\big(s^{(i)},\pi_k(s^{(i)})\big)\Big] + e'_1(N',\delta) \\
&\overset{(b)}{\leq} \frac{1}{N'}\sum_{i=1}^{N'}\Big[Q_{k-1}(s^{(i)},a^*) - \widehat{Q}_{k-1}\big(s^{(i)},\pi_k(s^{(i)})\big)\Big] + e'_1(N',\delta) + e'_2(N',M,\delta) && \text{w.p. } 1-2\delta' \\
&\overset{(c)}{\leq} \frac{1}{N'}\sum_{i=1}^{N'}\Big[Q_{k-1}(s^{(i)},a^*) - \widehat{Q}_{k-1}\big(s^{(i)},\tilde{\pi}(s^{(i)})\big)\Big] + e'_1(N',\delta) + e'_2(N',M,\delta) \\
&\overset{(b)}{\leq} \frac{1}{N'}\sum_{i=1}^{N'}\Big[Q_{k-1}(s^{(i)},a^*) - Q_{k-1}\big(s^{(i)},\tilde{\pi}(s^{(i)})\big)\Big] + e'_1(N',\delta) + 2e'_2(N',M,\delta) && \text{w.p. } 1-3\delta' \\
&= \mathcal{L}^{\Pi}_{k-1}(\widehat{\mu};\tilde{\pi}) + e'_1(N',\delta) + 2e'_2(N',M,\delta)
\end{aligned}$$

$$\begin{aligned}
&\overset{(a)}{\leq} \mathcal{L}^{\Pi}_{k-1}(\mu;\tilde{\pi}) + 2\big(e'_1(N',\delta) + e'_2(N',M,\delta)\big) \qquad \text{w.p. } 1-4\delta' \\
&= \inf_{\pi\in\Pi} \mathcal{L}^{\Pi}_{k-1}(\mu;\pi) + 2\big(e'_1(N',\delta) + e'_2(N',M,\delta)\big).
\end{aligned}$$

The statement of the theorem is obtained by setting $\delta' = \delta/4$.

(a) This follows from Lemma 19.
(b) Here we introduce the estimated action-value function $\widehat{Q}_{k-1}$ by bounding

$$\sup_{\pi\in\Pi}\left[\frac{1}{N'}\sum_{i=1}^{N'}\widehat{Q}_{k-1}\big(s^{(i)},\pi(s^{(i)})\big) - \frac{1}{N'}\sum_{i=1}^{N'}Q_{k-1}\big(s^{(i)},\pi(s^{(i)})\big)\right]$$

using Lemma 20.
(c) From the definition of π_k in CBMPI, we have

$$\pi_k = \operatorname*{argmin}_{\pi\in\Pi}\widehat{\mathcal{L}}^{\Pi}_{k-1}(\widehat{\mu};\pi) = \operatorname*{argmax}_{\pi\in\Pi}\frac{1}{N'}\sum_{i=1}^{N'}\widehat{Q}_{k-1}\big(s^{(i)},\pi(s^{(i)})\big),$$

thus, $-1/N'\sum_{i=1}^{N'}\widehat{Q}_{k-1}\big(s^{(i)},\pi_k(s^{(i)})\big)$ can be maximized by replacing π_k with any other policy, particularly with

$$\tilde{\pi} = \operatorname*{argmin}_{\pi\in\Pi}\int_{\mathcal{S}}\left(\max_{a\in\mathcal{A}}Q_{k-1}(s,a) - Q_{k-1}\big(s,\pi(s)\big)\right)\mu(ds).$$

■

References

Antos, A., Munos, R., and Szepesvári, C. (2007). Fitted Q-iteration in continuous action-space MDPs. In *Proceedings of the Advances in Neural Information Processing Systems 19*, pages 9–16.

Bertsekas, D. and Ioffe, S. (1996). Temporal differences-based policy iteration and applications in neuro-dynamic programming. Technical report, MIT.

Bertsekas, D. and Tsitsiklis, J. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

Burgiel, H. (1997). How to Lose at Tetris. *Mathematical Gazette*, **81**, 194–200.

Canbolat, P. and Rothblum, U. (2012). (Approximate) iterated successive approximations algorithm for sequential decision processes. *Annals of Operations Research*, pages 1–12.

Chang, C. and Lin, C. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, **2**(3), 27:1–27:27.

Demaine, E., Hohenberger, S., and Liben-Nowell, D. (2003). Tetris is hard, even to approximate. In *Proceedings of the Ninth International Computing and Combinatorics Conference*, pages 351–363.

Dimitrakakis, C. and Lagoudakis, M. (2008). Rollout sampling approximate policy iteration. *Machine Learning Journal*, **72**(3), 157–171.

Ernst, D., Geurts, P., and Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, **6**, 503–556.

Fahey, C. (2003). Tetris AI, Computer plays Tetris. `http://colinfahey.com/tetris/tetris.html`.

Farahmand, A., Munos, R., and Szepesvári, C. (2010). Error propagation for approximate policy and value iteration. In *Proceedings of the Advances in Neural Information Processing Systems 22*, pages 568–576.

Farias, V. and Van Roy, B. (2006). *Tetris: A study of randomized constraint sampling*. Springer-Verlag.

Fern, A., Yoon, S., and Givan, R. (2006). Approximate Policy Iteration with a Policy Language Bias: Solving Relational Markov Decision Processes. *Journal of Artificial Intelligence Research*, **25**, 75–118.

Furmston, T. and Barber, D. (2012). A unifying perspective of parametric policy search methods for Markov decision processes. In *Proceedings of the Advances in Neural Information Processing Systems 24*, pages 2726–2734.

Gabillon, V., Lazaric, A., Ghavamzadeh, M., and Scherrer, B. (2011). Classification-based policy iteration with a critic. In *Proceedings of the Twenty-Eighth International Conference on Machine Learning*, pages 1049–1056.

Gabillon, V., Ghavamzadeh, M., and Scherrer, B. (2013). Approximate dynamic programming finally performs well in the game of Tetris. In *Proceedings of Advances in Neural Information Processing Systems 26*, pages 1754–1762.

Geist, M. and Scherrer, B. (2014). Off-policy Learning with Eligibility Traces: A Survey. *Journal of Machine Learning Research*, **14**.

Gordon, G. (1995). Stable Function Approximation in Dynamic Programming. In *ICML*, pages 261–268.

Györfi, L., Kolher, M., Krzyżak, M., and Walk, H. (2002). *A Distribution-Free Theory of Nonparametric Regression*. Springer-Verlag.

Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, **9**, 159–195.

Kakade, S. (2002). A natural policy gradient. In *Proceedings of the Advances in Neural Information Processing Systems 14*, pages 1531–1538.

Kearns, M., Mansour, Y., and Ng, A. (2000). Approximate Planning in Large POMDPs via Reusable Trajectories. In *Proceedings of the Advances in Neural Information Processing Systems 12*, pages 1001–1007. MIT Press.

Lagoudakis, M. and Parr, R. (2003a). Least-Squares Policy Iteration. *Journal of Machine Learning Research*, **4**, 1107–1149.

Lagoudakis, M. and Parr, R. (2003b). Reinforcement Learning as Classification: Leveraging Modern Classifiers. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 424–431.

Lazaric, A., Ghavamzadeh, M., and Munos, R. (2010a). Analysis of a Classification-based Policy Iteration Algorithm. In *Proceedings of the Twenty-Seventh International Conference on Machine Learning*, pages 607–614.

Lazaric, A., Ghavamzadeh, M., and Munos, R. (2010b). Analysis of a classification-based policy iteration algorithm. Technical Report inria-00482065, INRIA.

Lazaric, A., Ghavamzadeh, M., and Munos, R. (2010c). Finite-sample analysis of LSTD. In *Proceedings of the Twenty-Seventh International Conference on Machine Learning*, pages 615–622.

Lazaric, A., Ghavamzadeh, M., and Munos, R. (2012). Finite-sample analysis of least-squares policy iteration. *Journal of Machine Learning Research*, **13**, 3041–3074.

Lesner, B. and Scherrer, B. (2013). Tight performance bounds for approximate modified policy iteration with non-stationary policies. *CoRR*, **abs/1304.5610**.

Munos, R. (2003). Error Bounds for Approximate Policy Iteration. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 560–567.

Munos, R. (2007). Performance Bounds in L_p-norm for Approximate Value Iteration. *SIAM J. Control and Optimization*, **46**(2), 541–561.

Munos, R. and Szepesvári, C. (2008). Finite-Time Bounds for Fitted Value Iteration. *Journal of Machine Learning Research*, **9**, 815–857.

Precup, D., Sutton, R., and Singh, S. (2000). Eligibility Traces for Off-Policy Policy Evaluation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 759–766.

Precup, D., Sutton, R., and Dasgupta, S. (2001). Off-Policy Temporal Difference Learning with Function Approximation. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 417–424.

Puterman, M. (1994). *Markov Decision Processes*. Wiley, New York.

Rubinstein, R. and Kroese, D. (2004). *The cross-entropy method: A unified approach to combinatorial optimization, Monte-Carlo simulation, and machine learning*. Springer-Verlag.

Scherrer, B. (2013). Performance Bounds for λ-Policy Iteration and Application to the Game of Tetris. *Journal of Machine Learning Research*, **14**, 1175–1221.

Scherrer, B. and Thiéry, C. (2010). Performance bound for approximate optimistic policy iteration. Technical report, INRIA.

Scherrer, B., Ghavamzadeh, M., Gabillon, V., and Geist, M. (2012). Approximate modified policy iteration. In *Proceedings of the Twenty Ninth International Conference on Machine Learning*, pages 1207–1214.

Singh, S. and Yee, R. (1994). An Upper Bound on the Loss from Approximate Optimal-Value Functions. *Machine Learning*, **16-3**, 227–233.

Szepesvári, C. (2010). Reinforcement Learning Algorithms for MDPs. In *Wiley Encyclopedia of Operations Research*. Wiley.

Szita, I. and Lőrincz, A. (2006). Learning Tetris Using the Noisy Cross-Entropy Method. *Neural Computation*, **18**(12), 2936–2941.

Thiery, C. and Scherrer, B. (2009a). Building Controllers for Tetris. *International Computer Games Association Journal*, **32**, 3–11.

Thiery, C. and Scherrer, B. (2009b). Improvements on Learning Tetris with Cross Entropy. *International Computer Games Association Journal*, **32**.

Thiery, C. and Scherrer, B. (2010a). Least-squares λ-policy iteration: Bias-variance trade-off in control problems. In *Proceedings of the Twenty-Seventh International Conference on Machine Learning*, pages 1071–1078.

Thiery, C. and Scherrer, B. (2010b). MDPTetris features documentation. `http://mdptetris.gforge.inria.fr/doc/feature_functions_8h.html`.

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

Tsitsiklis, J. and Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, **42**(5), 674–690.