HAL
open science

Approximate Policy Iteration Schemes: A Comparison

Bruno Scherrer

To cite this version:

Bruno Scherrer. Approximate Policy Iteration Schemes: A Comparison. ICML - 31st International Conference on Machine Learning - 2014, Jun 2014, Pékin, China. hal-00989982

HAL Id: hal-00989982

https://inria.hal.science/hal-00989982v1

Submitted on 12 May 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.

Approximate Policy Iteration Schemes: A Comparison

Bruno Scherrer BRUNO.SCHERRER@INRIA.FR
Inria, Villers-lès-Nancy, F-54600, France
Université de Lorraine, LORIA, UMR 7503, Vandœuvre-lès-Nancy, F-54506, France

Abstract

We consider the infinite-horizon discounted optimal control problem formalized by Markov Decision Processes. We focus on several approximate variations of the Policy Iteration algorithm: Approximate Policy Iteration (API) (Bertsekas & Tsitsiklis, 1996), Conservative Policy Iteration (CPI) (Kakade & Langford, 2002), a natural adaptation of the Policy Search by Dynamic Programming algorithm (Bagnell et al., 2003) to the infinite-horizon case (PSDP$_\infty$), and the recently proposed Non-Stationary Policy Iteration (NSPI(m)) (Scherrer & Lesner, 2012). For all algorithms, we describe performance bounds with respect the per-iteration error ϵ, and make a comparison by paying a particular attention to the concentrability constants involved, the number of iterations and the memory required. Our analysis highlights the following points: 1) The performance guarantee of CPI can be arbitrarily better than that of API, but this comes at the cost of a relative—exponential in $\frac{1}{\epsilon}$—increase of the number of iterations. 2) PSDP$_\infty$ enjoys the best of both worlds: its performance guarantee is similar to that of CPI, but within a number of iterations similar to that of API. 3) Contrary to API that requires a constant memory, the memory needed by CPI and PSDP$_\infty$ is proportional to their number of iterations, which may be problematic when the discount factor γ is close to 1 or the approximation error ϵ is close to 0; we show that the NSPI(m) algorithm allows to make an overall trade-off between memory and performance. Simulations with these schemes confirm our analysis.

Proceedings of the 31st International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32. Copyright 2014 by the author(s).

1. Introduction

We consider an infinite-horizon discounted Markov Decision Process (MDP) (Puterman, 1994; Bertsekas & Tsitsiklis, 1996) $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ is a possibly infinite state space, $\mathcal{A}$ is a finite action space, $P(ds'|s,a)$, for all (s,a), is a probability kernel on $\mathcal{S}$, $r : \mathcal{S} \to [-R_{\max}, R_{\max}]$ is a reward function bounded by $R_{\max}$, and $\gamma \in (0,1)$ is a discount factor. A stationary deterministic policy $\pi : \mathcal{S} \to \mathcal{A}$ maps states to actions. We write $P_\pi(ds'|s) = P(ds'|s, \pi(s))$ for the stochastic kernel associated to policy π. The value v_π of a policy π is a function mapping states to the expected discounted sum of rewards received when following π from these states: for all $s \in \mathcal{S}$,

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

The value v_π is clearly bounded by $V_{\max} = R_{\max}/(1-\gamma)$. It is well-known that v_π can be characterized as the unique fixed point of the linear Bellman operator associated to a policy π: $T_\pi : v \mapsto r + \gamma P_\pi v$. Similarly, the Bellman optimality operator $T : v \mapsto \max_\pi T_\pi v$ has as unique fixed point the optimal value $v_* = \max_\pi v_\pi$. A policy π is greedy w.r.t. a value function v if $T_\pi v = Tv$, the set of such greedy policies is written $\mathcal{G}v$. Finally, a policy π_* is optimal, with value $v_{\pi_*} = v_*$, iff $\pi_* \in \mathcal{G}v_*$, or equivalently $T_{\pi_*} v_* = v_*$.

The goal of this paper is to study and compare several approximate Policy Iteration schemes. In the literature, such schemes can be seen as implementing an approximate greedy operator, $\mathcal{G}_\epsilon$, that takes as input a distribution ν and a function $v : S \to \mathbb{R}$ and returns a policy π that is (ϵ, ν)-approximately greedy with respect to v in the sense that:

$$\nu(Tv - T_\pi v) = \nu(\max_{\pi'} T_{\pi'} v - T_\pi v) \leq \epsilon. \qquad (1)$$

where for all x, νx denotes $\mathbb{E}_{s\sim\nu}[x(s)]$. In practice, this approximation of the greedy operator can be achieved through a ℓ_p-regression of the so-called *Q-function*—the state-action value function—(a direct regression is suggested by Kakade & Langford (2002), a fixed-point LSTD approach is used by Lagoudakis & Parr (2003b)) or through a

(cost-sensitive) classification problem (Lagoudakis & Parr, 2003a; Lazaric et al., 2010). With this operator in hand, we shall describe several Policy Iteration schemes in Section 2. Then Section 3 will provide a detailed comparative analysis of their performance guarantees, time complexities, and memory requirements. Section 4 will go on by providing experiments that will illustrate their behavior, and confirm our analysis. Finally, Section 5 will conclude and present future work.

2. Algorithms

API We begin by describing the standard Approximate Policy Iteration (API) (Bertsekas & Tsitsiklis, 1996). At each iteration k, the algorithm switches to the policy that is approximately greedy with respect to the value of the previous policy for some distribution ν:

$$\pi_{k+1} \leftarrow \mathcal{G}_{\epsilon_{k+1}}(\nu, v_{\pi_k}). \tag{2}$$

If there is no error ($\epsilon_k = 0$) and ν assigns a positive weights to every state, it can easily be seen that this algorithm generates the same sequence of policies as exact Policy Iterations since from Equation (1) the policies are exactly greedy.

CPI/CPI(α)/API(α) We now turn to the description of Conservative Policy Iteration (CPI) proposed by (Kakade & Langford, 2002). At iteration k, CPI (described in Equation (3)) uses the distribution $d_{\pi_k,\nu} = (1-\gamma)\nu(I - \gamma P_{\pi_k})^{-1}$—the discounted cumulative occupancy measure induced by π_k when starting from ν—for calling the approximate greedy operator, and uses a stepsize α_k to generate a stochastic mixture of all the policies that are returned by the successive calls to the approximate greedy operator, which explains the adjective "conservative":

$$\pi_{k+1} \leftarrow (1-\alpha_{k+1})\pi_k + \alpha_{k+1}\mathcal{G}_{\epsilon_{k+1}}(d_{\pi_k,\nu}, v_{\pi_k}) \tag{3}$$

The stepsize α_{k+1} can be chosen in such a way that the above step leads to an improvement of the expected value of the policy given that the process is initialized according to the distribution ν (Kakade & Langford, 2002). The original article also describes a criterion for deciding whether to stop or to continue. Though the adaptive stepsize and the stopping condition allows to derive a nice analysis, they are in practice conservative: the stepsize α_k should be implemented with a line-search mechanism, or be fixed to some small value α. We will refer to this latter variation of CPI as CPI(α).

It is natural to also consider the algorithm API(α) (mentioned by Lagoudakis & Parr (2003a)), a variation of API that is conservative like CPI(α) in the sense that it mixes the new policy with the previous ones with weights α and $1-\alpha$, but that directly uses the distribution ν in the approximate greedy step:

$$\pi_{k+1} \leftarrow (1-\alpha)\pi_k + \alpha\mathcal{G}_{\epsilon_{k+1}}(\nu, v_{\pi_k}) \tag{4}$$

Because it uses ν instead of $d_{\pi_k,\nu}$, API(α) is simpler to implement than CPI(α)[1].

PSDP$_\infty$ We are now going to describe an algorithm that has a flavour similar to API—in the sense that at each step it does a full step towards a new deterministic policy—but also has a conservative flavour like CPI—in the sense that the policies considered evolve more and more slowly. This algorithm is a natural variation of the Policy Search by Dynamic Programming algorithm (PSDP) of Bagnell et al. (2003), originally proposed to tackle finite-horizon problems, to the infinite-horizon case; we thus refer to it as PSDP$_\infty$. To the best of our knowledge however, this variation has never been used in an infinite-horizon context.

The algorithm is based on finite-horizon non-stationary policies. Given a sequence of stationary deterministic policies (π_k) that the algorithm will generate, we will write $\sigma_k = \pi_k\pi_{k-1}\dots\pi_1$ the k-horizon policy that makes the first action according to π_k, then the second action according to π_{k-1}, etc. Its value is $v_{\sigma_k} = T_{\pi_k}T_{\pi_{k-1}}\dots T_{\pi_1}r$. We will write $\varnothing$ the "empty" non-stationary policy. Note that $v_\varnothing = r$ and that any infinite-horizon policy that begins with $\sigma_k = \pi_k\pi_{k-1}\dots\pi_1$, which we will (somewhat abusively) denote "$\sigma_k\dots$" has a value $v_{\sigma_k\dots} \geq v_{\sigma_k} - \gamma^k V_{\max}$. Starting from $\sigma_0 = \varnothing$, the algorithm implicitely builds a sequence of non-stationary policies (σ_k) by iteratively concatenating the policies that are returned by the approximate greedy operator:

$$\pi_{k+1} \leftarrow \mathcal{G}_{\epsilon_{k+1}}(\nu, v_{\sigma_k}) \tag{5}$$

While the standard PSDP algorithm of Bagnell et al. (2003) considers a horizon T and makes T iterations, the algorithm we consider here has an *indefinite* number of iterations. The algorithm can be stopped at any step k. The theory that we are about to describe suggests that one may return any policy that starts by the non-stationary policy σ_k. Since σ_k is an approximately good finite-horizon policy, and as we consider an infinite-horizon problem, a natural output that one may want to use in practice is the infinite-horizon policy that loops over σ_k, that we shall denote $(\sigma_k)^\infty$.

[1] In practice, controlling the greedy step with respect to $d_{\pi_k,\nu}$ requires to generate samples from this very distribution. As explained by Kakade & Langford (2002), one such sample can be done by running one trajectory starting from ν and following π_k, stopping at each step with probability $1-\gamma$. In particular, one sample from $d_{\pi_k,\nu}$ requires on average $\frac{1}{1-\gamma}$ samples from the underlying MDP. With this respect, API(α) is much simpler to implement.

From a practical point of view, PSDP$_\infty$ and CPI need to store all the (stationary deterministic) policies generated from the start. The memory required by the algorithmic scheme is thus proportional to the number of iterations, which may be prohibitive. The aim of the next paragraph, that presents the last algorithm of this article, is to describe a solution to this potential memory issue.

NSPI(m) We originally devised the algorithmic scheme of Equation (5) (PSDP$_\infty$) as a simplified variation of the *Non-Stationary PI algorithm with a growing period* algorithm (NSPI-growing) (Scherrer & Lesner, 2012)[2]. With respect to Equation (5), the only difference of NSPI-growing resides in the fact that the approximate greedy step is done with respect to the value $v_{(\sigma_k)^\infty}$ of the policy that loops infinitely over σ_k (formally the algorithm does $\pi_{k+1} \leftarrow \mathcal{G}_{\epsilon_{k+1}}(\nu, v_{(\sigma_k)^\infty})$) instead of the value v_{σ_k} of only the first k steps here. Following the intuition that when k is big, these two values will be close to each other, we ended up considering PSDP$_\infty$ because it is simpler. NSPI-growing suffers from the same memory drawback as CPI and PSDP$_\infty$. Interestingly, the work of Scherrer & Lesner (2012) contains another algorithm, *Non-Stationary PI with a fixed period* (NSPI(m)), that has a parameter that directly controls the number of policies stored in memory.

Similarly to PSDP$_\infty$, NSPI(m) is based on non-stationary policies. It takes as an input a parameter m. It requires a set of m initial deterministic stationary policies $\pi_{m-1}, \pi_{m-2}, \ldots, \pi_0$ and iteratively generates new policies $\pi_1, \pi_2, \ldots$. For any $k \geq 0$, we shall denote σ_k^m the m-horizon non-stationary policy that runs *in reverse order* the last m policies, which one may write formally: $\sigma_k^m = \pi_k \, \pi_{k-1} \, \ldots \, \pi_{k-m+1}$. Also, we shall denote $(\sigma_k^m)^\infty$ the m-periodic infinite-horizon non-stationary policy that loops over σ_k^m. Starting from $\sigma_0^m = \pi_0 \pi_1 \ldots \pi_{m-1}$, the algorithm iterates as follows:

$$\pi_{k+1} \leftarrow \mathcal{G}_{\epsilon_{k+1}}(\nu, v_{(\sigma_k^m)^\infty}) \tag{6}$$

Each iteration requires to compute an approximate greedy policy π_{k+1} with respect to the value $v_{(\sigma_k^m)^\infty}$ of $(\sigma_k^m)^\infty$, that is the fixed point of the compound operator[3]:

$$\forall v, \; T_{k,m} v = T_{\pi_k} T_{\pi_{k-1}} \ldots T_{\pi_{k-m+1}} v.$$

When one goes from iterations k to $k+1$, the process consists in adding π_{k+1} at the front of the $(m-1)$-horizon policy $\pi_k \pi_{k-1} \ldots \pi_{k-m+2}$, thus forming a new m-horizon policy σ_{k+1}^m. Doing so, we forget about the oldest policy π_{k-m+1} of σ_k^m and keep a constant memory of size m. At any step k, the algorithm can be stopped, and the output is the policy $\pi_{k,m} = (\sigma_k^m)^\infty$ that loops on σ_k^m. It is easy to see that NSPI(m) reduces to API when $m = 1$. Furthermore, if we assume that the reward function is positive, add "stop actions" in every state of the model that lead to a terminal absorbing state with a null reward, and initialize with an infinite sequence of policies that only take this "stop action", then NSPI(m) with $m = \infty$ reduces to PSDP$_\infty$.

3. Analysis

For all considered algorithms, we are going to describe bounds on the expected loss $E_{s\sim\mu}[v_{\pi_*}(s) - v_\pi(s)] = \mu(v_{\pi_*} - v_\pi)$ of using the (possibly stochastic or non-stationary) policy π ouput by the algorithms instead of the optimal policy π_* from some initial distribution μ of interest as a function of an upper bound ϵ on all errors (ϵ_k). In order to derive these theoretical guarantees, we will first need to introduce a few concentrability coefficients that relate the distribution μ with which one wants to have a guarantee, and the distribution ν used by the algorithms[4].

Definition 1. *Let $c(1), c(2), \ldots$ be the smallest coefficients in $[1,\infty) \cup \{\infty\}$ such that for all i and all sets of deterministic stationary policies $\pi_1, \pi_2, \ldots, \pi_i$, $\mu P_{\pi_1} P_{\pi_2} \ldots P_{\pi_i} \leq c(i)\nu$. For all m, k, we define the following coefficients in $[1,\infty) \cup \{\infty\}$:*

$$C^{(1,k)} = (1-\gamma) \sum_{i=0}^{\infty} \gamma^i c(i+k),$$

$$C^{(2,m,k)} = (1-\gamma)(1-\gamma^m) \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} \gamma^{i+jm} c(i+jm+k).$$

Similarly, let $c_{\pi_*}(1), c_{\pi_*}(2), \ldots$ be the smallest coefficients in $[1,\infty) \cup \{\infty\}$ such that for all i, $\mu (P_{\pi_*})^i \leq c_{\pi_*}(i)\nu$. We define:

$$C_{\pi_*}^{(1)} = (1-\gamma) \sum_{i=0}^{\infty} \gamma^i c_{\pi_*}(i).$$

Finally let C_{π_*} be the smallest coefficient in $[1,\infty) \cup \{\infty\}$ such that $d_{\pi_*,\mu} = (1-\gamma)\mu(I - \gamma P_{\pi_*})^{-1} \leq C_{\pi_*}\nu$.

With these notations in hand, our first contribution is to provide a thorough comparison of all the algorithms. This is done in Table 1. For each algorithm, we describe some performance bounds and the required number of iterations and memory. To make things clear, we only display the dependence with respect to the concentrability constants, the

[2] We later realized that it was in fact a very natural variation of PSDP. To "give Caesar his due and God his", we kept as the main reference the older work and gave the name PSDP$_\infty$.

[3] Implementing this algorithm in practice can trivially be done through cost-sensitive classification in a way similar to Lazaric et al. (2010). It could also be done with a straight-forward extension of LSTD(λ) to non-stationary policies.

[4] The expected loss corresponds to some weighted ℓ_1-norm of the loss $v_{\pi_*} - v_\pi$. Relaxing the goal to controlling the weighted ℓ_p-norm for some $p \geq 2$ allows to introduce some finer coefficients (Farahmand et al., 2010; Scherrer et al., 2012). Due to lack of space, we do not consider this here.

Algorithm	Performance Bound			# Iter.	Memory	Reference
API (Eq. (2))	$C^{(2,1,0)}$	$\frac{1}{(1-\gamma)^2}$	ϵ	$\frac{1}{1-\gamma}\log\frac{1}{\epsilon}$	1	(Lazaric et al., 2010)
(= NSPI(1))	$C^{(1,0)}$	$\frac{1}{(1-\gamma)^2}$	$\epsilon\log\frac{1}{\epsilon}$			
API(α) (Eq. (4)	$C^{(1,0)}$	$\frac{1}{(1-\gamma)^2}$	ϵ	$\frac{1}{\alpha(1-\gamma)}\log\frac{1}{\epsilon}$		
CPI(α)	$C^{(1,0)}$	$\frac{1}{(1-\gamma)^3}$	ϵ	$\frac{1}{\alpha(1-\gamma)}\log\frac{1}{\epsilon}$		
CPI (Eq. (3))	$C^{(1,0)}$	$\frac{1}{(1-\gamma)^3}$	$\epsilon\log\frac{1}{\epsilon}$	$\frac{1}{1-\gamma}\frac{1}{\epsilon}\log\frac{1}{\epsilon}$		
	C_{π_*}	$\frac{1}{(1-\gamma)^2}$	ϵ	$\frac{\gamma}{\epsilon^2}$		(Kakade & Langford, 2002)
PSDP$_\infty$ (Eq. (5))	C_{π_*}	$\frac{1}{(1-\gamma)^2}$	$\epsilon\log\frac{1}{\epsilon}$	$\frac{1}{1-\gamma}\log\frac{1}{\epsilon}$		
($\simeq$ NSPI(∞))	$C^{(1)}_{\pi_*}$	$\frac{1}{1-\gamma}$	ϵ	$\frac{1}{1-\gamma}\log\frac{1}{\epsilon}$		
NSPI(m) (Eq. (6))	$C^{(2,m,0)}$	$\frac{1}{(1-\gamma)(1-\gamma^m)}$	ϵ	$\frac{1}{1-\gamma}\log\frac{1}{\epsilon}$	m	
	$\frac{C^{(1,0)}}{m}$	$\frac{1}{(1-\gamma)^2(1-\gamma^m)}$	$\epsilon\log\frac{1}{\epsilon}$	$\frac{1}{1-\gamma}\log\frac{1}{\epsilon}$		
	$C^{(1)}_{\pi_*}+\gamma^m\frac{C^{(2,m,m)}}{1-\gamma^m}$	$\frac{1}{1-\gamma}$	ϵ	$\frac{1}{1-\gamma}\log\frac{1}{\epsilon}$		
	$C_{\pi_*}+\gamma^m\frac{C^{(2,m,0)}}{m(1-\gamma^m)}$	$\frac{1}{(1-\gamma)^2}$	$\epsilon\log\frac{1}{\epsilon}$	$\frac{1}{1-\gamma}\log\frac{1}{\epsilon}$		

Table 1. **Upper bounds on the performance guarantees for the algorithms.** Except when references are given, the bounds are to our knowledge new. A comparison of API and CPI based on the two known bounds was done by Ghavamzadeh & Lazaric (2012). The first bound of NSPI(m) can be seen as an adaptation of that provided by Scherrer & Lesner (2012) for the more restrictive ℓ_∞-norm setting.

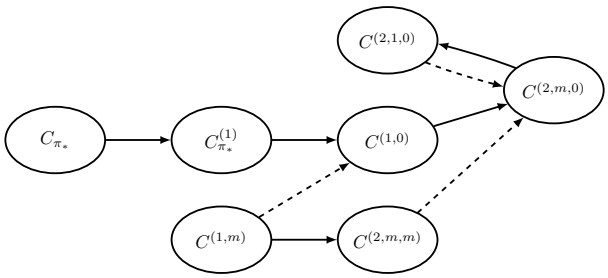

Figure 1. **Hierarchy of the concentrability constants.** A constant A is better than a constant B—see the text for details—if A is a parent of B on the above graph. The best constant is C_{π_*}.

discount factor γ, the quality ϵ of the approximate greedy operator, and—if applicable—the main parameters α/m of the algorithms. For API(α), CPI(α), CPI and PSDP$_\infty$, the required memory matches the number of iterations. All but two bounds are to our knowledge original. The derivation of the new results are given in Appendix A.

Our second contribution, that is complementary with the comparative list of bounds, is that we can show that there exists a hierarchy among the constants that appear in all the bounds of Table 1. In the directed graph of Figure 1, a constant B is a descendent of A *if and only if* the implication $\{B < \infty \Rightarrow A < \infty\}$ holds[5]. The "if and only if" is important here: it means that if A is a parent of B, and B is not a parent of A, then there exists an MDP for which A is finite while B is infinite; in other words, an algorithm that has a guarantee with respect to A has a guarantee that can be arbitrarily better than that with constant B. Thus, the overall best concentrability constant is C_{π_*}, while the worst are $C^{(2,1,0)}$ and $C^{(2,m,0)}$. To make the picture complete, we should add that for any MDP and any distribution μ, it is possible to find an input distribution ν for the algorithm (recall that the concentrability coefficients depend on ν and μ) such that C_{π_*} is finite, though it is not the case for $C^{(1)}_{\pi_*}$ (and as a consequence all the other coefficients). The derivation of this order relations is done in Appendix B.

The standard API algorithm has guarantees expressed in terms of $C^{(2,1,0)}$ and $C^{(1,0)}$ only. Since CPI's analysis can be done with respect to C_{π_*}, it has a performance guarantee that can be arbitrarily better than that of API, though the opposite is not true. This, however, comes at the cost of an exponential increase of time complexity since CPI may require a number of iterations that scales in $O\left(\frac{1}{\epsilon^2}\right)$, while the guarantee of API only requires $O\left(\log\frac{1}{\epsilon}\right)$ iterations. When the analysis of CPI is relaxed so that the performance guarantee is expressed in terms of the (worse) coefficient $C^{(1,0)}$ (obtained also for API), we can slightly improve the rate—to $\tilde{O}\left(\frac{1}{\epsilon}\right)$—, though it is still exponentially slower than that of API. This second result for CPI was proved with a technique that was also used for CPI(α) and API(α). We conjecture that it can be improved for CPI(α), that should be as good as CPI when α is sufficiently small.

PSDP$_\infty$ enjoys two guarantees that have a fast rate like those of API. One bound has a better dependency with respect to $\frac{1}{1-\gamma}$, but is expressed in terms of the worse coefficient $C^{(1)}_{\pi_*}$. The second guarantee is almost as good as that

[5]Dotted arrows are used to underline the fact that the comparison of coefficients is restricted to the case where the parameter m is finite.

of CPI since it only contains an extra $\log \frac{1}{\epsilon}$ term, but it has the nice property that it holds quickly with respect to ϵ: in time $O(\log \frac{1}{\epsilon})$ instead of $O(\frac{1}{\epsilon^2})$, that is exponentially faster. PSDP$_\infty$ is thus theoretically better than both CPI (as good but faster) and API (better and as fast).

Now, from a practical point of view, PSDP$_\infty$ and CPI need to store all the policies generated from the start. The memory required by these algorithms is thus proportional to the number of iterations. Even if PSDP$_\infty$ may require much fewer iterations than CPI, the corresponding memory requirement may still be prohibitive in situations where ϵ is small or γ is close to 1. We explained that NSPI(m) can be seen as making a bridge between API and PSDP$_\infty$. Since (i) both have a nice time complexity, (ii) API has the best memory requirement, and (iii) NSPI(m) has the best performance guarantee, NSPI(m) is a good candidate for making a standard performance/memory trade-off. If the first two bounds of NSPI(m) in Table 1 extends those of API, the other two are made of two terms: the left terms are identical to those obtained for PSDP$_\infty$, while the two possible right terms are new, but are controlled by γ^m, which can thus be made arbitrarily small by increasing the memory parameter m. Our analysis thus confirms our intuition that NSPI(m) allows to make a performance/memory trade-off in between API (small memory) and PSDP$_\infty$ (best performance). In other words, as soon as memory becomes a constraint, NSPI(m) is the natural alternative to PSDP$_\infty$.

4. Experiments

In this section, we present some experiments in order to illustrate the empirical behavior of the different algorithms discussed in the paper. We considered the standard API as a baseline. CPI, as it is described by Kakade & Langford (2002), is very slow (in one sample experiment on a 100 state problem, it made very slow progress and took several millions of iterations before it stopped) and we did not evaluate it further. Instead, we considered two variations: CPI+ that is identical to CPI except that it chooses the step α_k at each iteration by doing a line-search towards the policy output by the greedy operator[6], and CPI(α) with $\alpha = 0.1$, that makes "relatively but not too small" steps at each iteration. To assess the utility for CPI to use the distribution $d_{\nu,\pi}$ for the approximate greedy step, we also considered API(α) with $\alpha = 0.1$, the variation of API described in Equation (4) that makes small steps, and that only differs from CPI(α) by the fact that the approximate greedy step uses the distribution ν instead of $d_{\pi_k,\nu}$. In addition to these algorithms, we considered PSDP$_\infty$ and NSPI(m) for the values $m \in \{5, 10, 30\}$.

[6]We implemented a crude line-search mechanism, that looks on the set $2^i\alpha$ where α is the minimal step estimated by CPI to ensure improvement.

In order to assess their quality, we consider finite problems where the exact value function can be computed. More precisely, we consider Garnet problems first introduced by Archibald et al. (1995), which are a class of randomly constructed finite MDPs. They do not correspond to any specific application, but remain representative of the kind of MDP that might be encountered in practice. In brief, we consider Garnet problems with $|\mathcal{S}| \in \{50, 100, 200\}$, $|\mathcal{A}| \in \{2, 5, 10\}$ and branching factors in $\{1, 2, 10\}$. The greedy step used by all algorithms is approximated by an exact greedy operator applied to a noisy orthogonal projection on a linear space of dimension $\frac{|\mathcal{S}|}{10}$ with respect to the quadratic norm weighted by ν or $d_{\nu,\pi}$ (for CPI+ and CPI(α)) where ν is uniform.

For each of these $3^3 = 27$ parameter instances, we generated 30 i.i.d. Garnet MDPs $(M_i)_{1\leq i\leq 30}$. For each such MDP M_i, we ran API, API(0.1), CPI+, CPI(0.1), NSPI(m) for $m \in \{5, 10, 30\}$ and PSDP$_\infty$ 30 times. For each run j and algorithm, we compute for all iterations $k \in (1, 100)$ the performance, i.e. the loss $L_{j,k} = \mu(v_{\pi_*} - v_{\pi_k})$ with respect to the optimal policy. Figure 2 displays statistics about these random variables. For each algorithm, we display a learning curve with confidence regions that account for the variability across runs and problems. The supplementary material contains statistics that are respectively conditioned on the values of n_S, n_A and b, which gives some insight on the influence of these parameters.

From these experiments and statistics, we can make a series of observations. The standard API scheme is much more variable than the other algorithms and tends to provide the worst performance on average. CPI+ and CPI(α) display about the same asymptotic performance on average. If CPI(α) has slightly less variability, it is much slower than CPI+, that always converges in very few iterations (most of the time less than 10, and always less than 20). API(α)—the naive conservative variation of API that is also simpler than CPI(α)—is empirically close to CPI(α), while being on average slightly worse. CPI+, CPI(α) and PSDP$_\infty$ have a similar average performance, but the variability of PSDP$_\infty$ is significantly smaller. PSDP$_\infty$ is the algorithm that overall gives the best results. NSPI(m) does indeed provide a bridge between API and PSDP$_\infty$. By increasing m, the behavior gets closer to that of PSDP$_\infty$. With $m = 30$, NSPI(m) is overall better than API(α), CPI+, and CPI(α), and close to PSDP$_\infty$. The above relative observations are stable with respect to the number of states n_S and actions n_A. Interestingly, the differences between the algorithms tend to vanish when the dynamics of the problem gets more and more stochastic (when the branching factor increases). This complies with our analysis based on concentrability coefficients: there are all finite when the dynamics mixes a lot, and their relative difference are the biggest in deterministic instances.

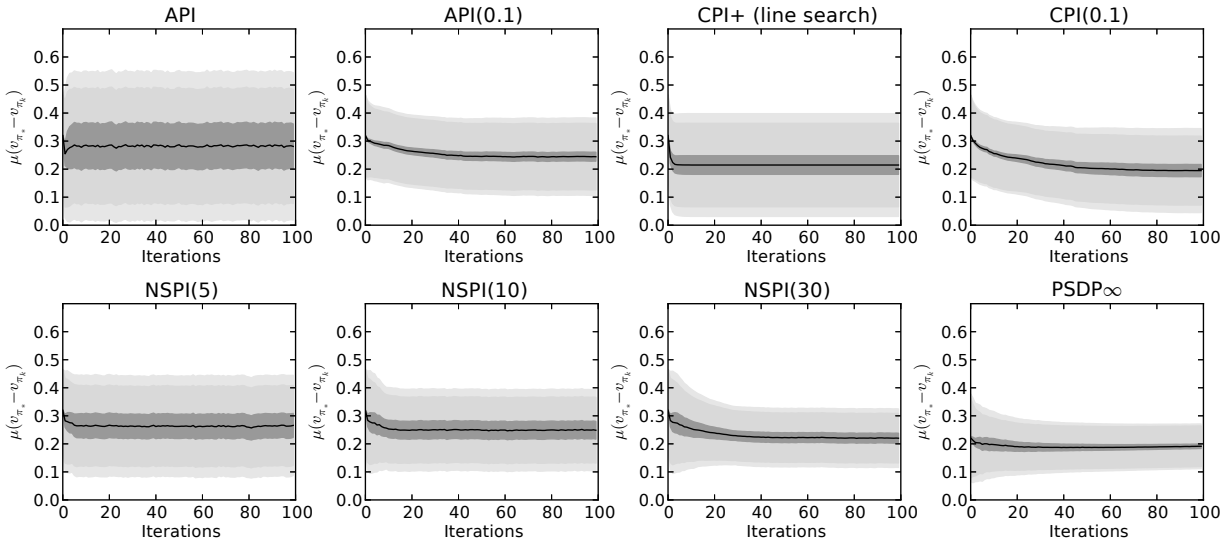

Figure 2. **Statistics for all instances.** The MDPs $(M_i)_{1 \leq i \leq 30}$ are i.i.d. with the same distribution as M_1. Conditioned on some MDP M_i and some algorithm, the error measures at all iteration k are i.i.d. with the same distribution as $L_{1,k}$. The central line of the learning curves gives the empirical estimate of the overall average error $(\mathbb{E}[L_{1,k}])_k$. The three grey regions (from dark to light grey) are estimates of respectively the variability (across MDPs) of the average error $(\text{Std}[\mathbb{E}[L_{1,k}|M_1]])_k$, the average (across MDPs) of the standard deviation of the error $(\mathbb{E}[\text{Std}[L_{1,k}|M_1]])_k$, and the variability (across MDPs) of the standard deviation of the error $(\text{Std}[\text{Std}[L_{1,k}|M_1]])_k$. For ease of comparison, all curves are displayed with the same x and y range.

5. Discussion, Summary and Future Work

We have considered several variations of the Policy Iteration schemes for infinite-horizon problems: API, CPI, NSPI(m), API(α) and PSDP$_\infty$[7]. We have in particular explained the fact—to our knowledge so far unknown—that the recently introduced NSPI(m) algorithm generalizes API (that is obtained when m=1) and PSDP$_\infty$ (that is very similar when $m = \infty$). Figure 1 synthesized the theoretical guarantees about these algorithms. Most of the bounds are to our knowledge new.

One of the first important message of our work is that what is usually hidden in the constants of the performance bounds does matter. The constants involved in the bounds for API, CPI, PSDP$_\infty$ and for the main (left) terms of NSPI(m) can be sorted from the worst to the best as follows: $C^{(2,1,0)}, C^{(1,0)}, C^{(1)}_{\pi_*}, C_{\pi_*}$. A detailed hierarchy of all constants was depicted in Figure 1. This is to our knowledge the first time that such an in-depth comparison of the bounds is done, and our hierarchy of constants has interesting implications that go beyond the Policy Iteration schemes we have been focusing on in this paper. As a matter of fact, several other dynamic programming algorithms, namely AVI (Munos, 2007), λPI (Scherrer, 2013), AMPI (Scherrer et al., 2012), come with guarantees involving the worst constant $C^{(2,1,0)}$, which suggests that they should not be competitive with the best algorithms we have described here.

At the purely technical level, several of our bounds come in pair; this is due to the fact that we have introduced a new proof technique. This led to a new bound for API, that improves the state of the art in the sense that it involves the constant $C^{(1,0)}$ instead of $C^{(2,1,0)}$. It also enabled us to derive new bounds for CPI (and its natural algorithmic variant CPI(α)) that is worse in terms of guarantee but has a better time complexity ($\tilde{O}(\frac{1}{\epsilon})$ instead of $O(\frac{1}{\epsilon^2})$). We believe this new technique may be helpful in the future for the analysis of other MDP algorithms.

Let us sum up the main insights of our analysis. 1) The guarantee for CPI can be arbitrarily stronger than that of API/API(α), because it is expressed with respect to the best concentrability constant C_{π_*}, but this comes at the cost of a relative—exponential in $\frac{1}{\epsilon}$—increase of the number of iterations. 2) PSDP$_\infty$ enjoys the best of both worlds: its performance guarantee is similar to that of CPI, but within a number of iterations similar to that of API. 3) Contrary to API that requires a constant memory, the memory needed by CPI and PSDP$_\infty$ is proportional to their number of iterations, which may be problematic in particular when the discount factor γ is close to 1 or the approximation error ϵ is close to 0; we showed that the NSPI(m) algorithm allows to make an overall trade-off between memory and perfor-

[7] We recall that to our knowledge, the use of PSDP$_\infty$ (PSDP in an *infinite-horizon* context) is not documented in the literature.

mance.

The main assumption of this work is that all algorithms have at disposal an ϵ-approximate greedy operator. It may be unreasonable to compare all algorithms on this basis, since the underlying optimization problems may have different complexities: for instance, methods like CPI look in a space of stochastic policies while API moves in a space of deterministic policies. Digging and understanding in more depth what is potentially hidden in the term ϵ—as we have done here for the concentrability constants—constitutes a very natural research direction.

Last but not least, we have run numerical experiments that support our worst-case analysis. On simulations on about 800 Garnet MDPs with various characteristics, CPI(α), CPI+ (CPI with a crude line-search mechanism), PSDP$_\infty$ and NSPI(m) were shown to always perform significantly better than the standard API. CPI+, CPI(α) and PSDP$_\infty$ performed similarly on average, but PSDP$_\infty$ showed much less variability and is thus the best algorithm in terms of overall performance. Finally, NSPI(m) allows to make a bridge between API and PSDP$_\infty$, reaching an overall performance close to that of PSDP$_\infty$ with a controlled memory. Implementing other instances of these algorithmic schemes, running and analyzing experiments on bigger domains constitutes interesting future work.

A. Proofs for Table 1

PSDP$_\infty$: For all k, we have

$$v_{\pi_*} - v_{\sigma_k} = T_{\pi_*} v_{\pi_*} - T_{\pi_*} v_{\sigma_{k-1}} + T_{\pi_*} v_{\sigma_{k-1}} - T_{\pi_k} v_{\sigma_{k-1}}$$

$$\leq \gamma P_{\pi_*}(v_\pi - v_{\sigma_{k-1}}) + e_k$$

where we defined $e_k = \max_{\pi'} T_{\pi'} v_{\sigma_{k-1}} - T_{\pi_k} v_{\sigma_{k-1}}$. As P_{π_*} is non negative, we deduce by induction:

$$v_{\pi_*} - v_{\sigma_k} \leq \sum_{i=0}^{k-1} (\gamma P_{\pi_*})^i e_{k-i} + \gamma^k V_{\max}.$$

By multiplying both sides by μ, using the definition of the coefficients $c_{\pi_*}(i)$ and the fact that $\nu e_j \leq \epsilon_j \leq \epsilon$, we get:

$$\begin{aligned} \mu(v_{\pi_*} - v_{\sigma_k}) &\leq \sum_{i=0}^{k-1} \mu(\gamma P_{\pi_*})^i e_{k-i} + \gamma^k V_{\max} \quad (7) \\ &\leq \sum_{i=0}^{k-1} \gamma^i c_{\pi_*}(i) \epsilon_{k-i} + \gamma^k V_{\max} \\ &\leq \left(\sum_{i=0}^{k-1} \gamma^i c_{\pi_*}(i) \right) \epsilon + \gamma^k V_{\max}. \end{aligned}$$

The bound with respect to $C^{(1)}_{\pi_*}$ is obtained by using the fact that $v_{\sigma_k \ldots} \geq v_{\sigma_k} - \gamma^k V_{\max}$ and taking $k \geq \left\lceil \frac{\log \frac{2V_{\max}}{\epsilon}}{1-\gamma} \right\rceil$.

Starting back in Equation (7) and using the definition of C_{π_*} (in particular the fact that for all i, $\mu(\gamma P_{\pi_*})^i \leq \frac{1}{1-\gamma} d_{\pi^*,\mu} \leq \frac{C_{\pi_*}}{1-\gamma}\nu$) and the fact that $\nu e_j \leq \epsilon_j$, we get:

$$\begin{aligned} \mu(v_{\pi_*} - v_{\sigma_k}) &\leq \sum_{i=0}^{k-1} \mu(\gamma P_{\pi_*})^i e_{k-i} + \gamma^k V_{\max} \\ &\leq \frac{C_{\pi_*}}{1-\gamma} \sum_{i=1}^{k} \epsilon_i + \gamma^k V_{\max} \end{aligned}$$

and the other bound is obtained by using the fact that $v_{\sigma_k \ldots} \geq v_{\sigma_k} - \gamma^k V_{\max}$, $\sum_{i=1}^k \epsilon_i \leq k\epsilon$, and considering the number of iterations $k = \left\lceil \frac{\log \frac{2V_{\max}}{\epsilon}}{1-\gamma} \right\rceil$.

API/NSPI(m): API is identical to NSPI(1), and its bounds are particular cases of the first two bounds for NSPI(m), so we only consider NSPI(m). By following the proof technique of Scherrer & Lesner (2012), writing $\Gamma_{k,m} = (\gamma P_{\pi_k})(\gamma P_{\pi_{k-1}}) \cdots (\gamma P_{\pi_{k-m+1}})$ and $e_{k+1} = \max_{\pi'} T_{\pi'} v_{\pi_{k,m}} - T_{\pi_{k+1}} v_{\pi_{k,m}}$, one can show that:

$$v_{\pi_*} - v_{\pi_{k,m}} \leq \sum_{i=0}^{k-1} (\gamma P_{\pi_*})^i (I - \Gamma_{k-i,m})^{-1} e_{k-i} + \gamma^k V_{\max}.$$

Multiplying both sides by μ (and observing that $e_k \geq 0$) and the fact that $\nu e_j \leq \epsilon_j \leq \epsilon$, we obtain:

$$\begin{aligned} &\mu(v_{\pi_*} - v_{\pi_k}) \\ &\leq \sum_{i=0}^{k-1} \mu(\gamma P_{\pi_*})^i (I - \Gamma_{k-i,m})^{-1} e_{k-i} + \gamma^k V_{\max} \quad (8) \\ &\leq \sum_{i=0}^{k-1} \left(\sum_{j=0}^{\infty} \gamma^{i+jm} c(i+jm) \epsilon_{k-i} \right) + \gamma^k V_{\max} \quad (9) \\ &\leq \sum_{i=0}^{k-1} \sum_{j=0}^{\infty} \gamma^{i+jm} c(i+jm) \epsilon + \gamma^k V_{\max}, \quad (10) \end{aligned}$$

which leads to the first bound by taking $k \geq \left\lceil \frac{\log \frac{2V_{\max}}{\epsilon}}{1-\gamma} \right\rceil$. Starting back on Equation (9), assuming for simplicity that $\epsilon_{-k} = 0$ for all $k \geq 0$, we get:

$$\begin{aligned} &\mu(v_{\pi_*} - v_{\pi_k}) - \gamma^k V_{\max} \\ &\leq \sum_{l=0}^{\lceil \frac{k-1}{m} \rceil} \sum_{h=0}^{m-1} \sum_{j=0}^{\infty} \gamma^{h+(l+j)m} c(h+(l+j)m) \epsilon_{k-h-lm} \\ &\leq \sum_{l=0}^{\lceil \frac{k-1}{m} \rceil} \sum_{h=0}^{m-1} \sum_{j=l}^{\infty} \gamma^{h+jm} c(h+jm) \max_{k-(l+1)m+1 \leq p \leq k-lm} \epsilon_p \\ &\leq \sum_{l=0}^{\lceil \frac{k-1}{m} \rceil} \sum_{h=0}^{m-1} \sum_{j=0}^{\infty} \gamma^{h+jm} c(h+jm) \max_{k-(l+1)m+1 \leq p \leq k-lm} \epsilon_p \end{aligned}$$

$$= \left(\sum_{h=0}^{m-1}\sum_{j=0}^{\infty}\gamma^{h+jm}c(h+jm)\right)\sum_{l=0}^{\lceil\frac{k-1}{m}\rceil}\max_{l-(l+1)m+1\leq p\leq k-lm}\epsilon_p$$

$$\leq \left(\sum_{i=0}^{\infty}\gamma^i c(i)\right)\left\lceil\frac{k-1}{m}\right\rceil\epsilon, \tag{11}$$

which leads to the second bound by taking $k = \left\lceil\frac{\log\frac{2V_{\max}}{\epsilon}}{1-\gamma}\right\rceil$. Last but not least, starting back on Equation (8), and using the fact that $(I-\Gamma_{k-i,m})^{-1} = I + \Gamma_{k-i,m}(I-\Gamma_{k-i,m})^{-1}$ we see that:

$$\mu(v_{\pi_*}-v_{\pi_k}) - \gamma^k V_{\max} \leq \sum_{i=0}^{k-1}\mu(\gamma P_{\pi_*})^i e_{k-i} + \\ + \sum_{i=0}^{k-1}\mu(\gamma P_{\pi_*})^i\Gamma_{k-i,m}(I-\Gamma_{k-i,m})^{-1}e_{k-i}.$$

The first term of the r.h.s. can be bounded exactly as for PSDP$_\infty$. For the second term, we have:

$$\sum_{i=0}^{k-1}\mu(\gamma P_{\pi_*})^i\Gamma_{k-i,m}(I-\Gamma_{k-i,m})^{-1}e_{k-i} \\ \leq \sum_{i=0}^{k-1}\sum_{j=1}^{\infty}\gamma^{i+jm}c(i+jm)\epsilon_{k-i} \\ = \gamma^m\sum_{i=0}^{k-1}\sum_{j=0}^{\infty}\gamma^{i+jm}c(i+(j+1)m)\epsilon_{k-i},$$

and we follow the same lines as above (from Equation (9) to Equations (10) and (11)) to conclude.

CPI, CPI(α), API(α): Conservative steps are addressed by a tedious generalization of the proof for API by Munos (2003). Due to lack of space, the proof is deferred to the Supplementary Material.

B. Proofs for Figure 1

We here provide details on the order relation for the concentrability coefficients.

$C_{\pi_*} \to C^{(1)}_{\pi_*}$**:** (i) We have $C_{\pi_*} \leq C^{(1)}_{\pi_*}$ because

$$d_{\pi_*,\mu} = (1-\gamma)\mu(I-\gamma P_{\pi_*})^{-1} = (1-\gamma)\sum_{i=0}^{\infty}\gamma^i\mu(P_{\pi_*})^i$$

$$\leq (1-\gamma)\sum_{i=0}^{\infty}\gamma^i c_{\pi_*}(i)\nu = C^{(1)}_{\pi_*}\nu$$

and C_{π_*} is the smallest coefficient C satisfying $d_{\pi_*,\mu} \leq C\nu$. (ii) We may have $C_{\pi_*} < \infty$ and $C^{(1)}_{\pi_*} = \infty$ by designing a MDP on $\mathbb{N}$ where π_* induces a deterministic transition from state i to state $i+1$.

$C^{(1)}_{\pi_*} \to C^{(1,0)}$**:** (i) We have $C^{(1)}_{\pi_*} \leq C^{(1,0)}$ because for all i, $c_{\pi_*}(i) \leq c(i)$. (ii) It is easy to obtain $C^{(1)}_{\pi_*} < \infty$ and $C^{(1,0)} = \infty$ since $C^{(1)}_{\pi_*}$ only depends on *one* policy while $C^{(1)}_{\pi_*}$ depends on *all* policies.

$C^{(1,0)} \to C^{(2,m,0)}$ **and** $C^{(1,m)} \to C^{(2,m,m)}$**:** (i) $C^{(1,m)} \leq \frac{1}{1-\gamma^m}C^{(2,m,m)}$ holds because

$$\frac{C^{(1,m)}}{1-\gamma} = \sum_{i=0}^{\infty}\gamma^i c(i+m) \leq \sum_{i=0}^{\infty}\sum_{j=0}^{\infty}\gamma^{i+jm}c(i+(j+1)m) \\ = \frac{1}{(1-\gamma)(1-\gamma^m)}C^{(2,m,m)}.$$

(ii) One may have $C^{(1,m)} < \infty$ and $C^{(2,m,m)} = \infty$ when $c(i) = \Theta(\frac{1}{i^2\gamma^i})$, since the generic term of $C^{(1,m)}$ is $\Theta(\frac{1}{i^2})$ (the sum converges) while that of $C^{(2,m,m)}$ is $\Theta(\frac{1}{i})$ (the sum diverges). The reasoning is similar for the other relation.

$C^{(1,m)} \to C^{(1,0)}$ **and** $C^{(2,m,m)} \to C^{(2,m,0)}$**:** We here assume that $m < \infty$. (i) We have $C^{(1,m)} \leq \frac{1}{\gamma^m}C^{(1,0)}$ and $C^{(2,m,m)} \leq \frac{1}{\gamma^m}C^{(2,m,0)}$. (ii) It suffices that $c(j) = \infty$ for some $j < m$ to have $C^{(2,m,0)} = \infty$ while $C^{(2,m,m)} < \infty$, or to have $C^{(1,0)} = \infty$ while $C^{(1,m)} < \infty$.

$C^{(2,1,0)} \leftrightarrow C^{(2,m,0)}$**:** (i) We clearly have $C^{(2,m,0)} \leq \frac{1-\gamma^m}{1-\gamma}C^{(2,1,0)}$. (ii) $C^{(2,m,0)}$ can be rewritten as follows:

$$C^{(2,m,0)} = (1-\gamma)(1-\gamma^m)\sum_{i=0}^{\infty}\left(1+\left\lfloor\frac{i}{m}\right\rfloor\right)\gamma^i c(i).$$

Then, using the fact that $1+\lfloor\frac{i}{m}\rfloor \geq \max\left(1,\frac{i}{m}\right)$, we have

$$\frac{1-\gamma}{1-\gamma^m}C^{(2,m,0)} \geq \sum_{i=0}^{\infty}\max\left(1,\frac{i}{m}\right)\gamma^i c(i) \\ \geq \sum_{i=0}^{m-1}\gamma^i c(i) + \sum_{i=m}^{\infty}\frac{i}{m}\gamma^i c(i) \\ \geq \sum_{i=0}^{m-1}\gamma^i c(i) + \frac{m}{m+1}\sum_{i=m}^{\infty}\frac{i+1}{m}\gamma^i c(i) \\ = \sum_{i=0}^{m-1}\gamma^i c(i) + \frac{m}{m+1}\left(C^{(2,1,0)} - \sum_{i=0}^{m-1}\gamma^i c(i)\right) \\ = \frac{m}{m+1}C^{(2,1,0)} + \frac{1}{m+1}\sum_{i=0}^{m-1}\gamma^i c(i).$$

Thus, when m is finite, $C^{(2,m,0)} < \infty \Rightarrow C^{(2,1,0)} < \infty$.

References

Archibald, T., McKinnon, K., and Thomas, L. On the Generation of Markov Decision Processes. *Journal of the Operational Research Society*, 46:354–361, 1995.

Bagnell, J.A., Kakade, S.M., Ng, A., and Schneider, J. Policy search by dynamic programming. In *NIPS*, 2003.

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

Farahmand, A.M., Munos, R., and Szepesvári, Cs. Error propagation for approximate policy and value iteration (extended version). In *NIPS*, 2010.

Ghavamzadeh, M. and Lazaric, A. Conservative and Greedy Approaches to Classification-based Policy Iteration. In *AAAI*, 2012.

Kakade, Sham and Langford, John. Approximately optimal approximate reinforcement learning. In *ICML*, 2002.

Lagoudakis, M. and Parr, R. Reinforcement Learning as Classification: Leveraging Modern Classifiers. In *ICML*, 2003a.

Lagoudakis, M.G. and Parr, R. Least-squares policy iteration. *Journal of Machine Learning Research (JMLR)*, 4: 1107–1149, 2003b.

Lazaric, A., Ghavamzadeh, M., and Munos, R. Analysis of a Classification-based Policy Iteration Algorithm. In *ICML*, 2010.

Munos, R. Error Bounds for Approximate Policy Iteration. In *ICML*, 2003.

Munos, R. Performance Bounds in Lp norm for Approximate Value Iteration. *SIAM J. Control and Optimization*, 2007.

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

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

Scherrer, B. and Lesner, B. On the Use of Non-Stationary Policies for Stationary Infinite-Horizon Markov Decision Processes. In *NIPS*, 2012.

Scherrer, Bruno, Ghavamzadeh, Mohammad, Gabillon, Victor, and Geist, Matthieu. Approximate Modified Policy Iteration. In *ICML*, 2012.

Supplementary Material

C. Proof for CPI, CPI(α), API(α)

We begin by proving the following result:

Theorem 1. *At each iteration $k < k^*$ of CPI (Equation (3)), the expected loss satisfies:*

$$\mu(v_{\pi_*} - v_{\pi_k}) \leq \frac{C^{(1,0)}}{(1-\gamma)^2} \sum_{i=1}^{k} \alpha_i \epsilon_i + e^{\{(1-\gamma)\sum_{i=1}^{k} \alpha_i\}} V_{\max}.$$

Proof. Using the facts that $T_{\pi_{k+1}} v_{\pi_k} = (1-\alpha_{k+1}) v_{\pi_k} + \alpha_{k+1} T_{\pi_{k+1}} v_{\pi_k}$ and the notation $e_{k+1} = \max_{\pi'} T_{\pi'} v_{\pi_k} - T_{\pi'_{k+1}} v_{\pi_k}$, we have:

$$\begin{aligned}
v_{\pi_*} - v_{\pi_{k+1}} &= v_{\pi_*} - T_{\pi_{k+1}} v_{\pi_k} + T_{\pi_{k+1}} v_{\pi_k} - T_{\pi_{k+1}} v_{\pi_{k+1}} \\
&= v_{\pi_*} - (1-\alpha_{k+1}) v_{\pi_k} - \alpha_{k+1} T_{\pi'_{k+1}} v_{\pi_k} + \gamma P_{\pi_{k+1}} (v_{\pi_k} - v_{\pi_{k+1}}) \\
&= (1-\alpha_{k+1})(v_{\pi_*} - v_{\pi_k}) + \alpha_{k+1}(T_{\pi_*} v_{\pi_*} - T_{\pi_*} v_{\pi_k}) + \alpha_{k+1}(T_{\pi_*} v_{\pi_k} - T_{\pi'_{k+1}} v_{\pi_k}) + \gamma P_{\pi_{k+1}} (v_{\pi_k} - v_{\pi_{k+1}}) \\
&\leq [(1-\alpha_{k+1}) I + \alpha_{k+1} \gamma P_{\pi_*}] (v_\pi - v_{\pi_k}) + \alpha_{k+1} e_{k+1} + \gamma P_{\pi_{k+1}} (v_{\pi_k} - v_{\pi_{k+1}}). \qquad (12)
\end{aligned}$$

Using the fact that $v_{\pi_{k+1}} = (I - \gamma P_{\pi_{k+1}})^{-1} r$, and the fact that $(I - \gamma P_{\pi_{k+1}})^{-1}$ is non-negative, we can see that

$$\begin{aligned}
v_{\pi_k} - v_{\pi_{k+1}} &= (I - \gamma P_{\pi_{k+1}})^{-1} (v_{\pi_k} - \gamma P_{\pi_{k+1}} v_{\pi_k} - r) \\
&= (I - \gamma P_{\pi_{k+1}})^{-1} (T_{\pi_k} v_{\pi_k} - T_{\pi_{k+1}} v_{\pi_k}) \\
&\leq (I - \gamma P_{\pi_{k+1}})^{-1} \alpha_{k+1} e_{k+1}.
\end{aligned}$$

Putting this back in Equation (12), we obtain:

$$v_{\pi_*} - v_{\pi_{k+1}} \leq [(1-\alpha_{k+1}) I + \alpha_{k+1} \gamma P_{\pi_*}] (v_\pi - v_{\pi_k}) + \alpha_{k+1} (I - \gamma P_{\pi_{k+1}})^{-1} e_{k+1}.$$

Define the matrix $Q_k = [(1-\alpha_k) I + \alpha_k \gamma P_{\pi_*}]$, the set $\mathcal{N}_{i,k} = \{j; k-i+1 \leq j \leq k\}$ (this set contains exactly i elements), the matrix $R_{i,k} = \prod_{j \in \mathcal{N}_{i,k}} Q_j$, and the coefficients $\beta_k = 1 - \alpha_k(1-\gamma)$ and $\delta_k = \prod_{i=1}^{k} \beta_k$. By repeatedly using the fact that the matrices Q_k are non-negative, we get by induction

$$v_{\pi_*} - v_{\pi_k} \leq \sum_{i=0}^{k-1} R_{i,k} \alpha_{k-i} (I - \gamma P_{\pi_{k-i}})^{-1} e_{k-i} + \delta_k V_{\max}. \qquad (13)$$

Let $\mathcal{P}_j(\mathcal{N}_{i,k})$ be the set of subsets of $\mathcal{N}_{i,k}$ of size j. With this notation we have

$$R_{i,k} = \sum_{j=0}^{i} \sum_{I \in \mathcal{P}_j(\mathcal{N}_{i,k})} \zeta_{I,i,k} (\gamma P_{\pi_*})^j$$

where for all subset I of $\mathcal{N}_{i,k}$, we wrote

$$\zeta_{I,i,k} = \left(\prod_{n \in I} \alpha_n \right) \left(\prod_{n \in \mathcal{N}_{i,k} \setminus I} (1 - \alpha_n) \right).$$

Therefore, by multiplying Equation (13) by μ, using the definition of the coefficients $c(i)$, and the facts that $\nu \leq (1 -$

$\gamma) d_{\nu,\pi_{k+1}}$, we obtain:

$$\begin{aligned}
\mu(v_{\pi_*} - v_{\pi_k}) &\leq \frac{1}{1-\gamma} \sum_{i=0}^{k-1} \sum_{j=0}^{i} \sum_{l=0}^{\infty} \sum_{I \in \mathcal{P}_j(\mathcal{N}_{i,k})} \zeta_{I,i,k} \gamma^{j+l} c(j+l) \alpha_{k-i} \epsilon_{k-i} + \delta_k V_{\max}. \\
&= \frac{1}{1-\gamma} \sum_{i=0}^{k-1} \sum_{j=0}^{i} \sum_{l=j}^{\infty} \sum_{I \in \mathcal{P}_j(\mathcal{N}_{i,k})} \zeta_{I,i,k} \gamma^{l} c(l) \alpha_{k-i} \epsilon_{k-i} + \delta_k V_{\max} \\
&\leq \frac{1}{1-\gamma} \sum_{i=0}^{k-1} \sum_{j=0}^{i} \sum_{l=0}^{\infty} \sum_{I \in \mathcal{P}_j(\mathcal{N}_{i,k})} \zeta_{I,i,k} \gamma^{l} c(l) \alpha_{k-i} \epsilon_{k-i} + \delta_k V_{\max} \\
&= \frac{1}{1-\gamma} \left(\sum_{l=0}^{\infty} \gamma^l c(l) \right) \sum_{i=0}^{k-1} \left(\sum_{j=0}^{i} \sum_{I \in \mathcal{P}_j(\mathcal{N}_{i,k})} \zeta_{I,i,k} \right) \alpha_{k-i} \epsilon_{k-i} + \delta_k V_{\max} \\
&= \frac{1}{1-\gamma} \left(\sum_{l=0}^{\infty} \gamma^l c(l) \right) \sum_{i=0}^{k-1} \left(\prod_{j \in \mathcal{N}_{i,k}} (1 - \alpha_j + \alpha_j) \right) \alpha_{k-i} \epsilon_{k-i} + \delta_k V_{\max} \\
&= \frac{1}{1-\gamma} \left(\sum_{l=0}^{\infty} \gamma^l c(l) \right) \left(\sum_{i=0}^{k-1} \alpha_{k-i} \epsilon_{k-i} \right) + \delta_k V_{\max}.
\end{aligned}$$

Now, using the fact that for $x \in (0,1)$, $\log(1-x) \leq -x$, we can observe that

$$\log \delta_k = \log \prod_{i=1}^{k} \beta_i = \sum_{i=1}^{k} \log \beta_i = \sum_{i=1}^{k} \log(1 - \alpha_i(1-\gamma)) \leq -(1-\gamma) \sum_{i=1}^{k} \alpha_i.$$

As a consequence, we get $\delta_k \leq e^{-(1-\gamma)\sum_{i=1}^{k} \alpha_i}$. □

In the analysis of CPI, Kakade & Langford (2002) show that the learning steps that ensure the nice performance guarantee of CPI satisfy $\alpha_k \geq \frac{(1-\gamma)\epsilon}{12\gamma V_{\max}}$, the right term $e^{\left\{(1-\gamma)\sum_{i=1}^{k} \alpha_i\right\}}$ above tends 0 exponentially fast, and we get the following corollary that shows that CPI has a performance bound with the coefficient $C^{(1,0)}$ of API in a number of iterations $O\left(\frac{\log \frac{1}{\epsilon}}{\epsilon}\right)$.

Corollary 1. *The smallest (random) iteration $k^\dagger$ such that $\frac{\log \frac{V_{\max}}{\epsilon}}{1-\gamma} \leq \sum_{i=1}^{k^\dagger} \alpha_i \leq \frac{\log \frac{V_{\max}}{\epsilon}}{1-\gamma} + 1$ is such that $k^\dagger \leq \frac{12\gamma V_{\max} \log \frac{V_{\max}}{\epsilon}}{\epsilon(1-\gamma)^2}$ and the policy $\pi_{k^\dagger}$ satisfies:*

$$\mu(v_{\pi_*} - v_{\pi_{k^\dagger}}) \leq \left(\frac{C^{(1,0)} \left(\sum_{i=1}^{k^\dagger} \alpha_i \right)}{(1-\gamma)^2} + 1 \right) \epsilon \leq \left(\frac{C^{(1,0)} \left(\log \frac{V_{\max}}{\epsilon} + 1 \right)}{(1-\gamma)^3} + 1 \right) \epsilon.$$

Since the proof is based on a generalization of the analysis of API and thus does not use any of the specific properties of CPI, it turns out that the results we have just given can straightforwardly be specialized to CPI(α).

Corollary 2. *Assume we run CPI(α) for some $\alpha \in (0,1)$, that is CPI (Equation (3)) with $\alpha_k = \alpha$ for all k.*

$$\textit{If } k = \left\lceil \frac{\log \frac{V_{\max}}{\epsilon}}{\alpha(1-\gamma)} \right\rceil, \quad \textit{then } \mu(v_{\pi_*} - v_{\pi_k}) \leq \frac{\alpha(k+1)C^{(1,0)}}{(1-\gamma)^2} \epsilon \leq \left(\frac{C^{(1,0)} \left(\log \frac{V_{\max}}{\epsilon} + 1 \right)}{(1-\gamma)^3} + 1 \right) \epsilon.$$

The above bound for CPI(α) involves the factor $\frac{1}{(1-\gamma)^3}$. A precise examination of the proof shows that this amplification is due to the fact that the approximate greedy operator uses the distribution $d_{\pi_k,\nu} \geq (1-\gamma)\nu$ instead of ν (for API). In fact, using a very similar proof, it is easy to show that API(α) satisfies the following result.

Corollary 3. *Assume API(α) is run for some* $\alpha \in (0,1)$.

$$\textit{If } k = \left\lceil \frac{\log \frac{V_{\max}}{\epsilon}}{\alpha(1-\gamma)} \right\rceil, \quad \textit{then } \mu(v_{\pi_*} - v_{\pi_k}) \leq \frac{\alpha(k+1)C^{(1,0)}}{(1-\gamma)}\epsilon \leq \left(\frac{C^{(1,0)}\left(\log \frac{V_{\max}}{\epsilon} + 1\right)}{(1-\gamma)^2} + 1 \right)\epsilon.$$

D. More details on the Numerical Simulations

Domain and Approximations In our experiments, a Garnet is parameterized by 4 parameters and is written $G(n_S, n_A, b, p)$: n_S is the number of states, n_A is the number of actions, b is a branching factor specifying how many possible next states are possible for each state-action pair (b states are chosen uniformly at random and transition probabilities are set by sampling uniform random $b-1$ cut points between 0 and 1) and p is the number of features (for linear function approximation). The reward is state-dependent: for a given randomly generated Garnet problem, the reward for each state is uniformly sampled between 0 and 1. Features are chosen randomly: Φ is a $n_S \times p$ feature matrix of which each component is randomly and uniformly sampled between 0 and 1. The discount factor γ is set to 0.99 in all experiments.

All the algorithms we have discussed in the paper need to repeatedly compute $\mathcal{G}_\epsilon(\rho, v)$ for some distribution $\rho = \nu$ or $\rho = d_{\pi,\nu}$. In other words, they must be able to make calls to an approximate greedy operator applied to the value v of some policy for some distribution ρ. To implement this operator, we compute a noisy estimate of the value v with a uniform white noise $u(\iota)$ of amplitude ι, then projects this estimate onto the space spanned by Φ with respect to the ρ-quadratic norm (projection that we write $\Pi_{\Phi,\rho}$), and then applies the (exact) greedy operator on this projected estimate. In a nutshell, one call to the approximate greedy operator $\mathcal{G}_\epsilon(\rho, v)$ amounts to compute $\mathcal{G}\Pi_{\Phi,\rho}(v + u(\iota))$.

Simulations We have run series of experiments, in which we callibrated the perturbations (noise, approximations) so that the algorithm are significantly perturbed but no too much (we do not want their behavior to become too erratic). After trial and error, we ended up considering the following setting. We used Garnet problems $G(n_S, n_A, b, p)$ with the number of states $n_S \in \{50, 100, 200\}$, the number of actions $n_A \in \{2, 5, 10\}$, the branching factor $b \in \{1, 2, 10\}\}$ ($b = 1$ corresponds to deterministic problems), the number of features to approximate the value $p = \frac{n_S}{10}$, and the noise level $\iota = 0.1$ (10%).

In addition to Figure 2 that shows the statistics overall for the all the parameter instances, Figure 3, 4 and 5 display statistics that are respectively conditioned on the values of n_S, n_A and b, which gives some insight on the influence of these parameters.

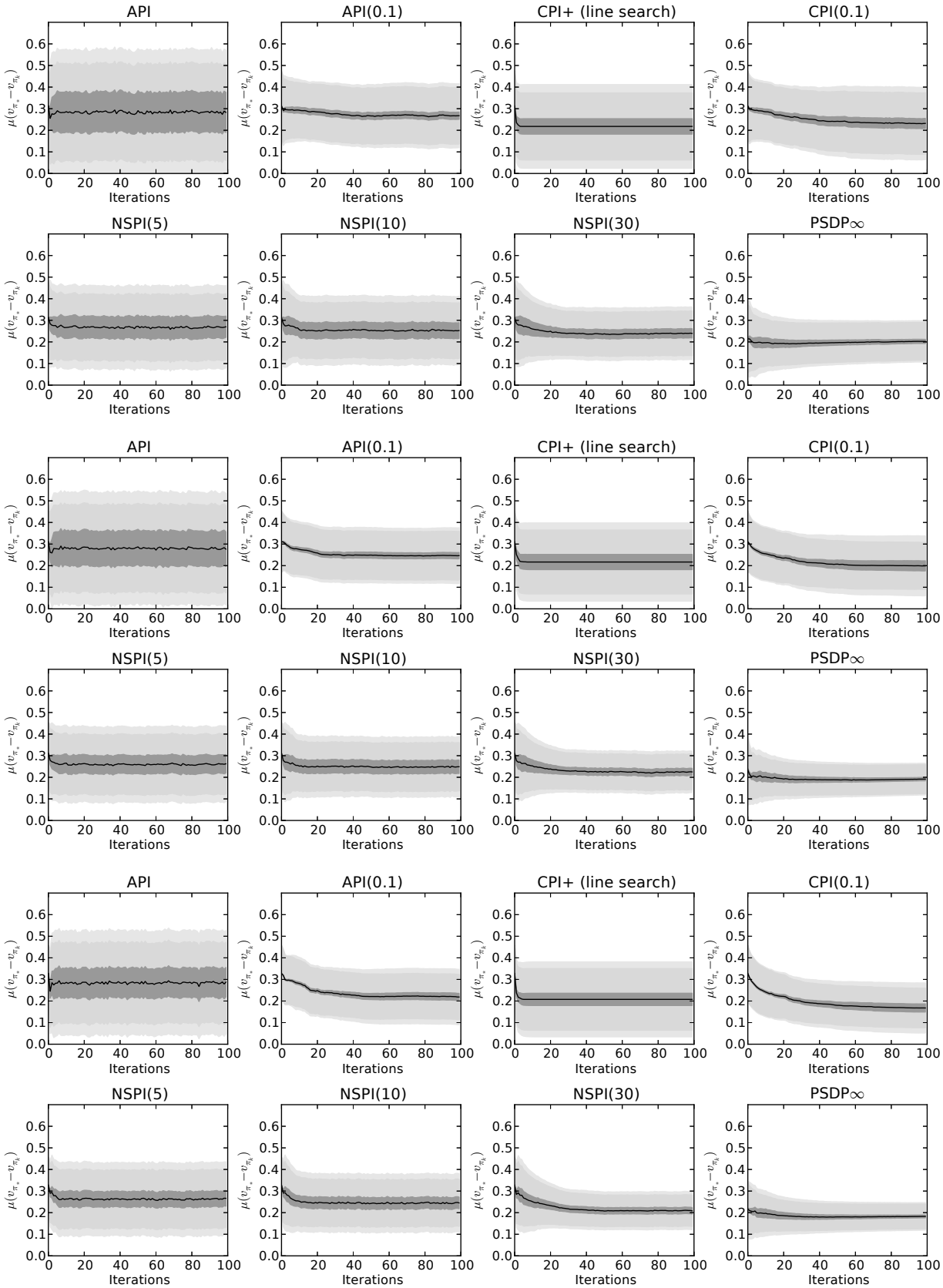

Figure 3. Statistics conditioned on the number of states. Top: $n_S = 50$. Middle: $n_S = 100$. Bottom $n_S = 200$.

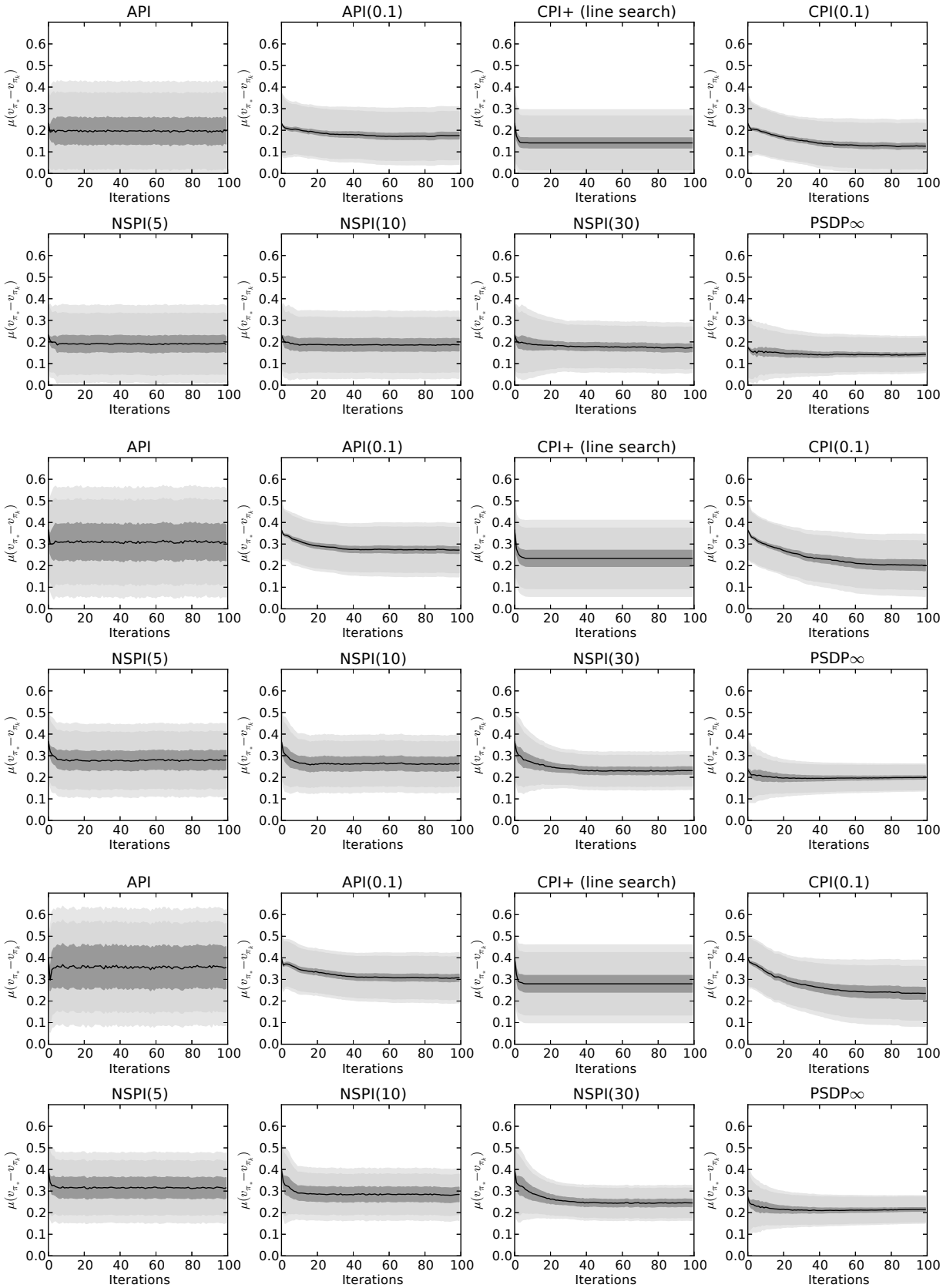

Figure 4. Statistics conditioned on the number of actions. Top: $n_A = 2$. Middle: $n_A = 5$. Bottom $n_a = 10$.

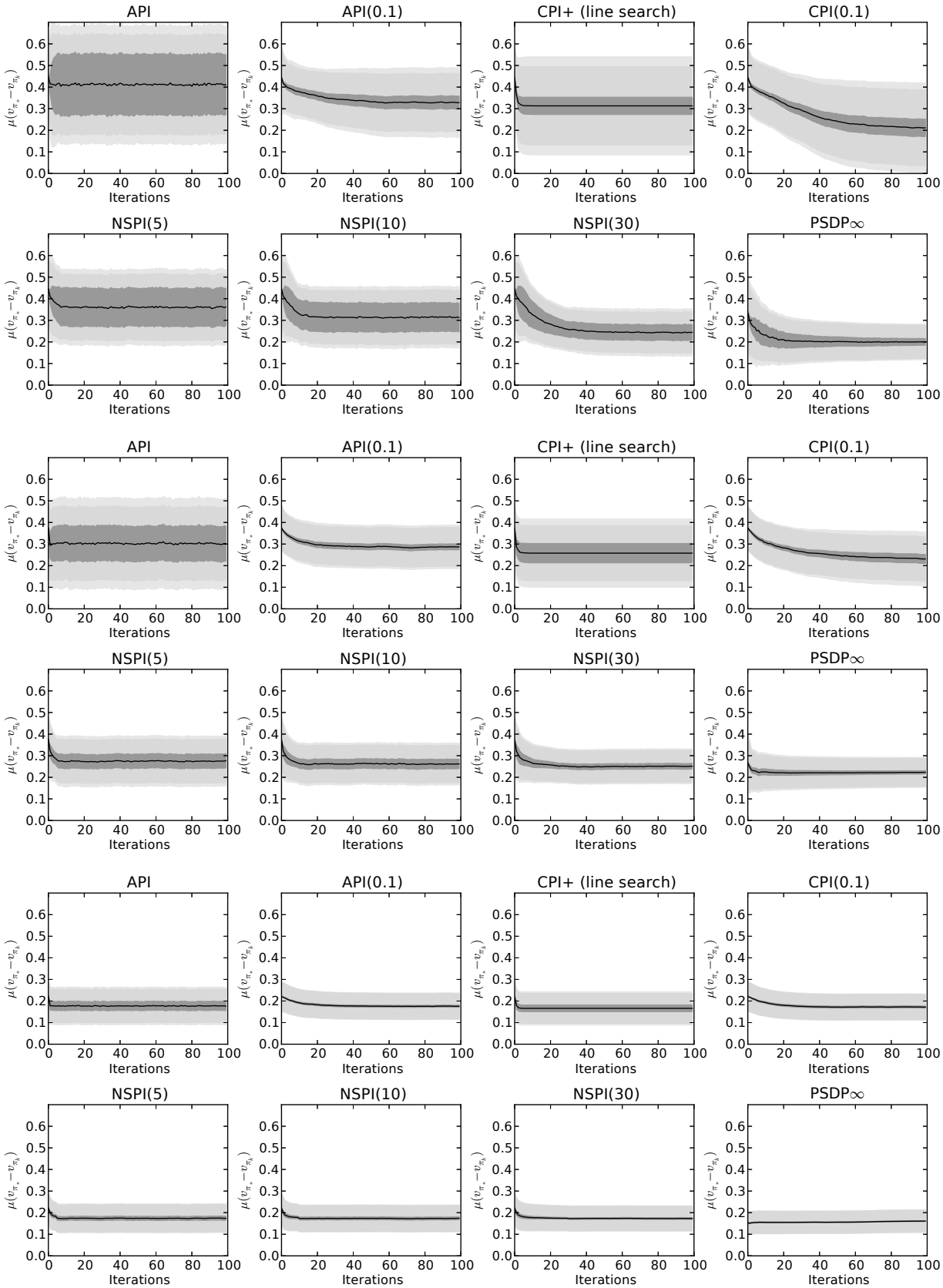

Figure 5. Statistics conditioned on the branching factor. Top: $b = 1$ (deterministic). Middle: $b = 2$. Bottom $b = 10$.