Une étude comparative de quelques schémas d'approximation de type iterations sur les politiques

Bruno Scherrer

Une étude comparative de quelques schémas d'approximation de type iterations sur les politiques

Bruno Scherrer

12 mai 2014

Résumé

Nous considérons le problème du contrôle optimal actualisé à horizon infini formalisé dans le cadre des processus de décision Markoviens. Nous nous focalisons sur plusieurs variations approchées du schéma itération sur les politiques : itérations sur les politiques approché (API) (Bertsekas and Tsitsiklis, 1996), itérations sur les politiques conservatif (CPI) (Kakade and Langford, 2002), une adaptation naturelle de l'algorithme "Policy Search by Dynamic Programming" (Bagnell *et al.*, 2003) au cas de l'horizon infini (PSDP_∞), et itérations sur les politiques non-stationnaires (NSPI(m)) (Scherrer and Lesner, 2012). Pour tous ces algorithmes, nous décrivons des bornes de performance en fonction de l'erreur ϵ à chaque itération, et faisons une comparaison en portant une attention particulière aux coefficients de concentration impliqués, au nombre d'itérations et à la mémoire requis. Notre analyse souligne plusieurs points : 1) Les garanties de performance de CPI peuvent être arbitrairement meilleures que celle d'API, mais au prix d'une augmentation—exponentielle en $\frac{1}{\epsilon}$—du nombre d'itérations. 2) PSDP_∞ combine les avantages d'API et CPI : sa garantie de performance est similaire à celle de CPI, et elle est obtenue en un nombre d'itérations identique à celui d'API. 3) Contrairement à API qui requiert une mémoire constante, la mémoire dont CPI et PSDP_∞ ont besoin est proportionnelle au nombre d'itérations, ce qui peut être problématique lorsque le facteur d'actualisation γ est proche de 1 ou lorsque l'erreur d'approximation ϵ est proche de 0 ; nous montrons que l'algorithme NSPI(m) permet de faire un compromis entre la bonne mémoire d'API et la meilleure performance de PSDP_∞. Enfin, des simulations numériques de ces schémas algorithmiques confirment notre analyse.

1 Introduction

Nous considérons un processus de décision Markovien (PDM) (Puterman, 1994; Bertsekas and Tsitsiklis, 1996) $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, où $\mathcal{S}$ est un espace d'états potentiellement infini, $\mathcal{A}$ un espace d'actions fini, $P(ds'|s,a)$, pour tout (s,a), est un noyau de transition sur $\mathcal{S}$, $r : \mathcal{S} \to [-R_{\max}, R_{\max}]$ est une fonction de récompense bornée par $R_{\max}$, et $\gamma \in (0,1)$ est un facteur d'actualisation. Une politique déterministe $\pi : \mathcal{S} \to \mathcal{A}$ associe une action à chaque état. Nous notons, pour tout état s, $P_\pi(ds'|s) = P(ds'|s,\pi(s))$ le noyau de transition associé à la politique π. La valeur v_π d'une politique π est une fonction associant à chaque état l'espérance de la somme actualisée des récompenses reçues en suivant π à partir de cet état : pour tout $s \in \mathcal{S}$,

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

La valeur v_π est clairement bornée par $V_{\max} = R_{\max}/(1-\gamma)$. Il est connu que v_π peut être caractérisée comme l'unique point fixe de l'opérateur de Bellman associé à la politique π : $T_\pi : v \mapsto r + \gamma P_\pi v$. De manière similaire, l'opérateur d'optimalité de Bellman $T : v \mapsto \max_\pi T_\pi v$ a pour unique point fixe la valeur optimale $v_* = \max_\pi v_\pi$. On dit qu'une politique π est gourmande par rapport à une fonction valeur v si $T_\pi v = Tv$, l'ensemble des politiques gourmandes étant noté $\mathcal{G}v$. Finalement, on sait qu'une politique π_* est optimale, ayant pour valeur $v_{\pi_*} = v_*$, si et seulement si $\pi_* \in \mathcal{G}v_*$, ou de manière équivalente $T_{\pi_*} v_* = v_*$.

L'objectif de cet article est d'étudier et de comparer plusieurs schémas approximatifs de type itérations sur les politiques. Dans la littérature, ces schémas peuvent être vus comme implémentant une approximation de l'opérateur gourmand, $\mathcal{G}_\epsilon$, qui prend en paramètres une distribution ν et une fonction

$v : S \to \mathbb{R}$, et renvoie une politique π qui est (ϵ, ν)-approximativement gourmande par rapport à v dans le sens suivant :

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

où pour tout x, νx est une notation pour $\mathbb{E}_{s\sim\nu}[x(s)]$. En pratique, cette approximation peut être effectuée via une régression ℓ_p de la fonction de valeur états/actions Q (une régression directe est suggérée pour CPI dans (Kakade and Langford, 2002; Kakade, 2003), une approche LSTD de type point fixe est utilisée pour LSPI (Lagoudakis and Parr, 2003a)) où via un problème de classification (à coût sensitif) (Lagoudakis and Parr, 2003b; Lazaric *et al.*, 2010). Avec cet opérateur, nous allons décrire plusieurs schémas de type itérations sur les politiques dans la Section 2. Ensuite, la Section 3 présentera une analyse détaillée et comparée de leur garanties de performance, de leur complexité en temps et en mémoire. La Section 4 présentera des simulations numériques qui illustrent leur comportement et confirme notre analyse. Finalement, la Section 5 concluera et présentera quelques perspectives.

2 Algorithmes

Itérations sur les politiques approché (API) Nous commençons par décrire le schéma le plus standard, itérations sur les politiques approché (API) (Bertsekas and Tsitsiklis, 1996). A chaque itération k, l'algorithme considère passe à la politique qui est approximativement gourmande par rapport à la valeur v_{π_k} de la politique courante π_k pour une distribution ν donnée :

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

S'il n'y a pas d'erreur ($\epsilon_k = 0$) et si ν met un poids positif en chaque état, il est facile de voir que cet algorithme génère la même séquence de politiques que la version exacte d'itérations sur les politiques, car d'après l'Equation (1), les politiques sont exactement gourmandes.

Itérations sur les politiques conservatif (CPI/CPI(α)/API(α)) Nous passons maintenant à la description d'itérations sur les politiques conservatif (CPI) proposé par Kakade and Langford (2002). A l'itération k, CPI (décrit par l'Equation (3)) utilise la distribution $d_{\pi_k,\nu} = (1-\gamma)\nu(I-\gamma P_{\pi_k})^{-1}$—la mesure d'occupation cumulée actualisée induite par π_k en partant de ν—pour appeler l'opérateur gourmand approché, et un pas d'apprentissage α_k pour générer un mélange stochastique de toutes les politiques qui ont été retournées par l'opérateur gourmand approché, ce qui explique l'adjectif "conservatif" :

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

Le pas d'apprentissage α_{k+1} peut être choisi de sorte que chaque itération induise une amélioration de l'espérance de la valeur de la politique sachant que le processus est initialisé selon ν (Kakade and Langford, 2002). L'article original décrit également un critère d'arrêt. Si l'utilisation d'un pas d'apprentissage adaptatif et de cette condition d'arrêt est intéressante pour dériver une analyse élégante de performance (voir (Kakade and Langford, 2002) et la prochaine section), elle est en général très conservatrice. En pratique, le pas d'apprentissage α_k doit être choisi par un mécanisme de recherche linéaire, ou fixé à une petite valeur α. Nous ferons référence à ce dernier cas de CPI en écrivant CPI(α).

Il est naturel de considérer également l'algorithme API(α) (évoqué par Lagoudakis and Parr (2003b)), qui est une variation d'API qui est conservative comme CPI(α) dans le sens où elle mélange la nouvelle politique avec les précédentes avec les poids α et $1 - \alpha$, mais qui utilise directement la distribution ν pour l'appel à l'opérateur approximativement gourmand :

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

Parce qu'il utilise ν au lieu de $d_{\pi_k,\nu}$, API(α) est plus simple à implémenter que CPI(α) [1].

1. En pratique, contrôler l'étape approximativement gourmande par rapport à $d_{\pi_k,\nu}$ recquiert de générer des échantillons selon cette distribution. Comme expliqué par Kakade and Langford (2002), un échantillon pour cette distribution peut être obtenu en simulant une trajectoire partant d'une distribution ν et suivant la politique π_k, et en s'arrêtant à chaque étape avec une probabilité $1-\gamma$. En particulier, la génération d'un échantillon pour $d_{\pi_k,\nu}$ recquiert en moyenne $\frac{1}{1-\gamma}$ échantillons du MDP sous-jacent. De ce point de vue, API(α) est beacoup plus simple à implémenter.

"Policy Search by Dynamic Programming" pour des problèmes à horizon infini (PSDP_∞) Nous allons à présent décrire un algorithme d'un type similaire à API—dans le sens où à chaque étape, il passe à une nouvelle politique déterministe—mais qui est conservatif comme CPI—dans le sens où les politiques considérées évoluent de plus en plus doucement. Cet algorithme est une variation naturelle de l'algorithme " Policy Search by Dynamic Programming" (PSDP) de Bagnell *et al.* (2003)—originellement proposé dans le cadre des problèmes à horizion fini—au cas de l'horion infini ; nous y ferons donc référence via l'appellation PSDP_∞. A notre connaissance cependant, cette variation n'a jamais été décrite dans la littérature.

L'algorithme est fondé sur l'utilisation de politiques non-stationnaires à horizon fini. Etant donnée une séquence de politiques déterministes stationnaires (π_k) que l'algorithme va progressivement générer, nous noterons $\sigma_k = \pi_k \pi_{k-1} \dots \pi_1$ la politique non-stationnaire à horizon k qui effectue la première action selon π_k, la deuxième selon π_{k-1}, etc. Sa valeur est $v_{\sigma_k} = T_{\pi_k} T_{\pi_{k-1}} \dots T_{\pi_1} r$. Nous noterons $\varnothing$ la politique non-stationnaire "vide". Remarquons que $v_\varnothing = r$ et que n'importe quelle politique à horizon infini qui commence par $\sigma_k = \pi_k \pi_{k-1} \dots \pi_1$, ce que nous noterons (abusivement) "$\sigma_k \dots$" a une valeur $v_{\sigma_k \dots} \geq v_\sigma - \gamma^k V_{\max}$. En partant de $\sigma_0 = \varnothing$, l'algorithme construit implicitement une séquence de politiques non-stationnaires (σ_k) en concaténant de manière itérative les politiques retournées par l'opérateur approximativement gourmand :

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

Tandis que l'algorithme originel PSDP de Bagnell *et al.* (2003) considère un horizon fini T et effectue T itérations, l'algorithme que nous considérons ici a un nombre *indéfini* d'itérations. L'algorithme peut être arrêté à n'importe quelle itération k. L'analyse que nous allons bientôt décrire suggère de retourner n'importe quelle politique qui commence par la politique non-stationnaire σ_k. Vu que σ_k est une politique à horizon fini qui est approximativment bonne, et que nous considérons le cas d'un problème à horizon infini, il est naturel de considérer en pratique comme résultat la politique qui boucle indéfiniment sur σ_k, ce que nous noterons $(\sigma_k)^\infty$.

D'un point de vue pratique, PSDP_∞ et CPI recquièrent de stocker toutes les politiques (déterministes stationnaires) générées depuis le début. La complexité en mémoire est donc proportionnelle au nombre d'itérations, ce qui peut s'avérer problématique. Le but du prochain paragraphe, qui présente le dernier algorithme étudié dans cet article, est de proposer une solution à ce problème potentiel de mémoire.

Itérations sur les politiques non-stationnaires (NSPI(m)) Nous avons originellement conçu le schéma algorithmique de l'Equation (5) (PSDP_∞) comme une simplification d'*itérations sur les politiques non-stationnaires avec une période croissante* (NSPI-croissant) récemment proposée par Scherrer and Lesner (2012) [2]. Par rapport à l'Equation (5), la seule différence de NSPI-croissant réside dans le fait que l'étape approximativement gourmande est faite par rapport à la valeur $v_{(\sigma_k)^\infty}$ de la politique qui répète indéfiniment σ_k (formellement l'algorithme fait $\pi_{k+1} \leftarrow \mathcal{G}_{\epsilon_{k+1}}(\nu, v_{(\sigma_k)^\infty})$) au lieu de la valeur v_{σ_k} des k premiers pas. Suivant l'intuition que lorsque k est grand, ces deux valeurs doivent être proches, nous avons considéré l'algorithme PSDP_∞ car il est plus simple.

NSPI-croissant souffre du même problème potentiel de mémoire que CPI et PSDP_∞. De manière intéressante, l'article de Scherrer and Lesner (2012) décrit un autre algorithme, itérations sur les politiques non-stationnaires avec une période fixe (NSPI(m)), qui prend en entrée un paramètre qui contrôle directement le nombre de politiques stockées en mémoire.

De manière similaire à PSDP_∞, NSPI(m) est fondé sur l'utilisation de politiques non-stationnaires. Il prend en entrée un paramètre m. Il recquiert un ensemble de m politiques déterministes stationnaires $\pi_{m-1}, \pi_{m-2}, \dots, \pi_0$ et génère itérativement de nouvelles politiques $\pi_1, \pi_2, \dots$. Pour tout $k \geq 0$, nous noterons σ_k^m la politique non-stationnaire à horizon fini m qui exécute *dans l'ordre inverse* les m dernières politiques, ce que nous écrirons formellement : $\sigma_k^m = \pi_k \ \pi_{k-1} \ \dots \ \pi_{k-m+1}$. De plus, nous noterons $(\sigma_k^m)^\infty$ la politique à horizon fini, non-stationnaire avec période m, qui boucle indéfiniment sur σ_k^m. Partant de $\sigma_0^m = \pi_0 \pi_1 \dots \pi_{m-1}$, l'algorithme itère de la façon suivante :

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

2. Nous avons pris conscience ultérieurement qu'il s'agissait d'une variation naturelle de PSDP. Pour "rendre à César ce qui est à César", nous avons gardé PSDP comme référence principale et donné le nom PSDP_∞.

Algorithme	Borne de l'erreur			# Iter.	Mémoire	Référence
API (Eq. (2)) (= NSPI(1))	$C^{(2,1,0)}$	$\frac{1}{(1-\gamma)^2}$	ϵ	$\frac{1}{1-\gamma}\log\frac{1}{\epsilon}$	1	(Lazaric *et al.*, 2010)
	$C^{(1,0)}$	$\frac{1}{(1-\gamma)^2}$	$\epsilon\log\frac{1}{\epsilon}$			
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 and Langford, 2002)
PSDP$_\infty$ (Eq. (5)) ($\simeq$ NSPI(∞))	C_{π_*}	$\frac{1}{(1-\gamma)^2}$	$\epsilon\log\frac{1}{\epsilon}$	$\frac{1}{1-\gamma}\log\frac{1}{\epsilon}$		
	$C^{(1)}_{\pi_*}$	$\frac{1}{1-\gamma}$	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 – Comparaison des algorithmes.

Chaque itération recquiert de calculer une politique π_{k+1} qui est approximativement gourmande par rapport à la valeur $v_{(\sigma_k^m)^\infty}$ of $(\sigma_k^m)^\infty$, qui est le point fixe de l'opérateur composé [3] :

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

Lorsqu'on passe de l'itération k à l'itération $k+1$, le processus consiste à ajouter la politique π_{k+1} au début de la politique $\pi_k\pi_{k-1}\dots\pi_{k-m+2}$ à horizon $m-1$, formant ainsi une nouvelle politique σ_{k+1}^m à horizon m. Ce faisant, l'algorithme oublie la politique la plus ancienne π_{k-m+1} de σ_k^m et garde une mémoire constante de taille m. A n'importe quelle itération, l'algorithme peut être stoppé, et on retourne la politique $\pi_{k,m} = (\sigma_k^m)^\infty$ à horizon infini qui répète indéfiniment σ_k^m. Il est facile de voir que NSPI(m) est identique à API lorsque $m = 1$. De manière plus intéressante, si on ajoute des actions spéciales "stop" en chaque état qui mène dans un état terminal avec une récompense strictement plus petite que $-R_{\max}$, et si on initialise avec une séquence infinie de politique qui ne prennent que cet action "stop", alors NSPI(m) avec $m = \infty$ est équivalent à PSDP$_\infty$.

3 Analyse

Pour tous les algorithmes considérés, nous allons décrire des bornes sur la perte $E_{s\sim\mu}[v_{\pi_*}(s) - v_\pi(s)] = \mu(v_{\pi_*} - v_\pi)$ liée au fait d'exécuter la politique π (potentiellement stochastique ou non-stationnaire) renvoyée par les algorithmes au lieu de la politique optimale π_* à partir d'une distribution μ, en fonction d'une borne uniforme ϵ sur les erreurs (ϵ_k). Afin de dériver ces garanties, nous aurons besoin d'introduire quelques constantes dites de concentrabilité, qui mesurent l'adéquation de la distribution μ avec laquelle on mesure la perte, et la distribution ν utilisée par les algorithmes.

Definition 1. *Soient $c(1), c(2), \dots$ les plus petits coefficients dans $[1,\infty)\cup\{\infty\}$ tels que pour tout i et tout ensemble de politiques déterministes stationnaires $\pi_1, \pi_2, \dots, \pi_i$, $\mu P_{\pi_1}P_{\pi_2}\dots P_{\pi_i} \le c(i)\nu$. Pour tout m, k, Nous définissons les constantes suivantes de $[1,\infty)\cup\{\infty\}$:*

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

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

3. En pratique, implémenter cet algorithme peut trivialement être fait via la classification à coût sensitif d'une manière similaire à celle suggérée dans Lazaric *et al.* (2010). Cela peut également être fait via une extension plutôt immédiate de LSTD au cas non-stationnaire.

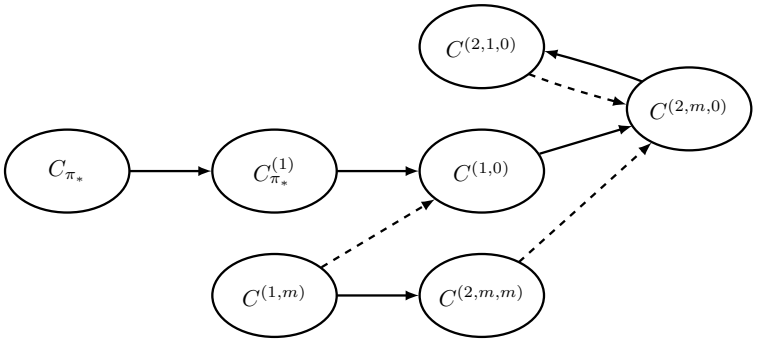

FIGURE 1 – Hiérarchie des constantes de concentrabilité

De manière similaire, soient $c_{\pi_*}(1), c_{\pi_*}(2), \ldots$ *les plus petits coefficients dans* $[1,\infty)\cup\{\infty\}$ *tels que pour tout* i, $\mu(P_{\pi_*})^i \leq c_{\pi_*}(i)\nu$. *Nous définissons la constante de* $[1,\infty)\cup\{\infty\}$ *:*

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

Finalement, soit C_{π_*} *le plus petite constante de* $[1,\infty)\cup\{\infty\}$ *tel que* $d_{\pi_*,\mu} = (1-\gamma)\mu(I-\gamma P_{\pi_*})^{-1} \leq C_{\pi_*}\nu$.

Avec ces notations, notre première contribution consiste à fournir une comparaison approfondie de tous les algorithmes. Ceci est fait Table 1. Pour chaque algorithme, nous décrivons des bornes de performances et le nombre d'itérations et la mémoire requis correspondants. Pour garder une présentation claire, nous montrons seulement la dépendance de ces quantités en fonction des coefficients de concentration, du facteur d'actualisation γ, de la qualité ϵ de l'opérateur gourmand approché, et—le cas échéant—du paramètre principal α/m des algorithmes. Pour API(α), CPI(α), CPI et PSDP$_\infty$, la mémoire requise égale le nombre d'itérations. La dernière colonne contient des références pour les bornes qui ne sont pas nouvelles. Toutes les bornes (sauf deux) sont à notre connaissance nouvelles. La dérivation des nouveaux résultats est donnée dans l'Appendix A.

Notre deuxième contribution, qui complète notre liste comparative de bornes, est que nous pouvons montrer qu'il existe une hiérarchie parmi les constantes qui apparaissent dans les bornes de la Table 1. Dans le graphe dirigé de la Figure 1, une constante B est une descendante de A *si et seulement si* l'implication $\{B < \infty \Rightarrow A < \infty\}$ est vraie [4]. L'équivalence "si et seulement si" est ici importante : elle signifie que si A est un parent de B, et B n'est pas un parent de A, alors il existe un PDM pour lequel A est finie tandis que B est infinie ; en d'autres termes, une garantie exprimée en fonction de la constante A peut être arbitrairement meilleure qu'une garantie exprimée en fonction de B. Ainsi, la meilleure constante de concentrabilité est C_{π_*}, tandis que les pires sont $C^{(2,1,0)}$ et $C^{(2,m,0)}$. Pour finir de compléter notre étude, ajoutons que pour tout PDM et toute distribution μ, il est possible de trouver une distribution ν pour l'algorithme (on rappelle que les coefficients de concentration dépendent de ν et μ) tel que C_{π_*} est fini, alors que ce n'est pas le cas pour $C^{(1)}_{\pi_*}$ (et par voie de conséquence pour toutes les autres constantes). La dérivation de cette relation d'ordre est effectuée dans l'Appendix B.

L'algorithme standard API jouit de garanties exprimées en fonction de $C^{(2,1,0)}$ et $C^{(1,0)}$ seulement. Comme CPI a une borne de performance en fonction de C_{π_*}, il a une garantie de performance qui peut être arbitrairement meilleure que celle d'API, tandis que le contraire n'est pas vrai. Ceci, cependant, se fait au prix d'une augmentation exponentielle de la complexité en temps, car CPI recquiert un nombre d'itération de l'ordre de $O(\frac{1}{\epsilon^2})$, tandis que la garantie d'API est obtenue après seulement $O\left(\log\frac{1}{\epsilon}\right)$ itérations. Quand l'analyse de CPI est relaxée de sorte à exprimer la borne de performance en fonction de la (moins bonne) constante $C^{(1,0)}$ (utilisée aussi pour API), nous pouvons légèrement améliorer la vitesse—$\tilde{O}\left(\frac{1}{\epsilon}\right)$—, bien qu'elle soit toujours exponentiellement moins bonne que celle d'API. Ce résultat est prouvé à l'aide d'une technique qui a aussi été utilisée pour CPI(α) et API(α). Nous conjecturons que la borne pour CPI(α) peut être améliorée, et notamment qu'elle doit être aussi bonne que celle de CPI dès lors que le coefficient d'apprentissage α est suffisamment petit.

4. Les flèches en ligne pointillée sont utilisées pour souligner le fait que la comparaison des coefficients est restreinte au cas où le paramètre m est fini.

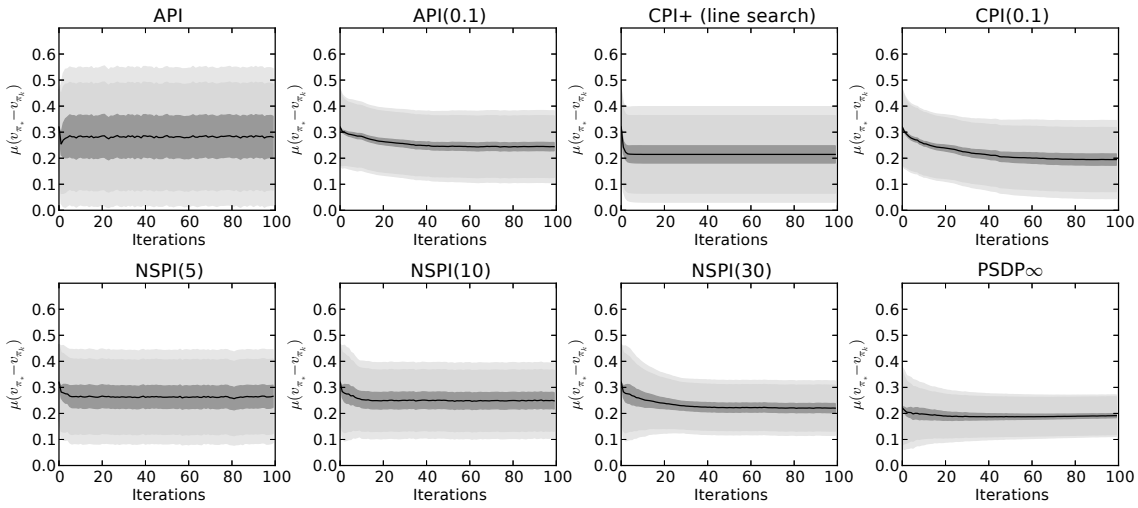

FIGURE 2 – **Statistiques pour toutes les instances.** Les PDMs $(M_i)_{1\leq i\leq 30}$ sont i.i.d. et ont la même distribution que M_1. Conditionnellement à un PDM M_i et un algorithme, les mesures d'erreur pour toutes les itérations k sont i.i.d. et ont la même distribution que $L_{1,k}$. La ligne centrale des courbes d'apprentissage donne un estimé empirique de la moyenne globale$(\mathbb{E}[L_{1,k}])_k$. Les trois régions grisées (du plus foncé au plus clair) sont respectivement des estimés de la variabilité (sur les PDMs) $(\mathbb{E}[\mathrm{Std}[L_{1,k}|M_1]])_k$ de l'erreur moyenne, la moyenne (sur les PDMs) $(\mathbb{E}[\mathrm{Std}[L_{1,k}|M_1]])_k$ de la déviation standard de l'erreur, et la variabilité (sur les PDMs) $(\mathrm{Std}[\mathrm{Std}[L_{1,k}|M_1]])_k$ de la déviation standard de l'erreur. Pour facilier les comparaisons, toutes les courbes sont affichés avec les même axes x et y

PSDP$_\infty$ jouit de deux garanties, et a une vitesse de convergence rapide comme celle d'API. Une borne a une meilleure dépendance par rapport à $\frac{1}{1-\gamma}$, mais est exprimée en fonction de la constante moins bonne $C^{(1)}_{\pi_*}$. La seconde garantie est presque aussi bonne que celle de CPI dans la mesure où elle contient seulement un terme $\log\frac{1}{\epsilon}$ en plus, mais a la propriété sympathique d'être garantie rapidement par rapport à ϵ : en un temps $O(\log\frac{1}{\epsilon})$ au lieu de $O(\frac{1}{\epsilon^2})$, c'est-à-dire exponentiellement plus vite. Ainsi, PSDP$_\infty$ est un algorithme qui est en théorie meilleur que CPI (aussi bon mais plus rapide) et API (aussi rapide mais meilleur).

Maintenant, d'un point de vue pratique, PSDP et CPI doivent stocker toutes les politiques générées depuis le début. La mémoire requise par ces algorithmes est ainsi proportionnelle au nombre d'itérations. Même si PSDP$_\infty$ a besoin d'un nombre d'itérations bien moindre que CPI, le besoin de mémoire correspondant peut être problématique lorsque ϵ est petit ou γ est proche de 1. Nous avons expliqué que NSPI(m) peut être vu comme généralisant API et PSDP$_\infty$. Comme (i) les deux ont une bonne complexité en temps, (ii) API a la meilleure complexité en mémoire, et (iii) PSDP$_\infty$ a la meilleure garantie de performance, NSPI(m) est un bon candidat pour faire un compromis entre la garantie et la mémoire. Si, dans la Table 1, les deux premières bornes pour cet algorithme étendent celle d'API, les deux suivantes sont composées de deux termes : les termes de gauche sont identiques à ceux obtenus pour PSDP$_\infty$, tandis que les termes de droite, nouveaux, sont contrôlés par γ^m, qui peut être rendu arbitrairement petit en augmentant le paramètre de mémoire m. Notre analyse confirme ainsi l'intuition que NSPI(m) permet de faire un compromis performance/mémoire entre API (plus petite mémoire) et PSDP$_\infty$ (meilleure performance). En d'autres termes, dès lors que la mémoire devient une contrainte, NSPI(m) constitue l'alternative naturelle à PSDP$_\infty$.

4 Simulations Numériques

Dans cette section, nous présentons des simulations numériques pour illustrer le comportement empirique des différents algorithmes considérés dans l'article. Nous considérons le schéma standard API comme référence. CPI, tel qu'il est décrit dans Kakade and Langford (2002), est très lent (sur une expé-

rience impliquant un problème avec 100 états, il a progressé très lentement et a mis plusieurs millions d'itérations pour s'arrêter) et nous ne l'avons pas plus évalué. A la place, nous avons considéré deux variations : CPI+ qui est identique à CPI sauf qu'il choisit un pas d'apprentissage α_k à chaque itération en faisant une recherche linéaire dans la direction de la politique fournie par l'opérateur gourmand approché [5], et CPI(α) with $\alpha = 0.1$, qui effectue des pas "relativement petits mais pas trop" à chaque itération. Afin d'évaluer l'utilité pour CPI d'utiliser la distribution $d_{\nu,\pi}$ lors du choix de la politique approximativement gourmande, nous avons considéré également API(α) avec $\alpha = 0.1$, la variation conservative d'API décrite à l'Equation (4) qui fait des petits pas, et qui diffère seulement de CPI(α) par le fait que l'opérateur approximativement gourmand utilise la distribution ν au lieu de $d_{\pi_k,\nu}$. En plus de ces algorithmes, nous considérons PSDP$_\infty$, et NSPI(m) avec les valeurs $m \in \{5, 10, 30\}$.

Afin de mesurer précisément leur qualité, nous considérons des PDMs de taille finie pour lesquels la fonction de la valeur optimale peut être calculée exactement. Plus précisément, nous considérons des problèmes "Garnet" introduits par Archibald *et al.* (1995), qui sont une classe de PDMs finis construits aléatoirement. Ils ne correspondent à aucune application particulière, mais sont représentatifs de PDMs que l'on peut rencontrer en pratique. En bref, nous considérons des problèmes Garnet avec $|\mathcal{S}| \in \{50, 100, 200\}$, $|\mathcal{A}| \in \{2, 5, 10\}$, et un facteur de branchement dans $\{1, 2, 10\}$. L'opérateur gourmand approché utilisé par tous les algorithmes est implémenté comme l'opérateur gourmand exact appliqué à une projection bruitée de la vraie fonction de valeur sur un espace linéaire de dimension $\frac{|\mathcal{S}|}{10}$. Cette projection est une projection orthogonale par rapport à la norme quadratique pondérée par ν ou $d_{\nu,\pi}$ (pour CPI+ et CPI(α)) où ν est uniforme.

Pour chacune des $3^3 = 27$ instances des paramètres de problèmes Garnet, nous générons 30 PDMs Garnet i.i.d. $(M_i)_{1 \leq i \leq 30}$. Pour chaque PDM M_i, nous exécutons API, API(0.1), CPI+, CPI(0.1), NSPI(m) pour $m \in \{5, 10, 30\}$ et PSDP$_\infty$ 30 fois. Pour chaque exécution, nous calculons, à chaque itération $k \in (1, 100)$ la performance, c'est-à-dire la perte $L_{j,k} = \mu(v_{\pi_*} - v_{\pi_k})$ par rapport à la politique optimale. La Figure 2 montre des statistiques sur ces variables aléatoires. Pour chaque algorithme, nous traçons des courbes d'apprentissage avec des intervalles de confiance qui rendent compte de la variabilité de la performance à travers les différentes exécutions et les différents problèmes tirés aléatoirement. Dans la partie "Compléments" de cet article, nous présentons des statistiques qui sont conditionnées aux valeurs de n_S, n_A et b, ce qui éclaire sur l'influence de ces paramètres.

A partir de ces expériences et des statistiques, nous pouvons faire une série d'observations. L'algorithme standard API est bien plus variable que les autres et tend à fournir les pires performances en moyenne. CPI+ et CPI(α) ont une performance asymptotique moyenne similaire. Si CPI(α) montre un peu moins de variabilité, il est beaucoup plus lent que CPI+ qui converge toujours en un petit nombre d'itérations (la plupart du temps en moins de 10 itérations, et toujours en moins de 20). API(α)—la variation conservative naïve d'API qui est aussi plus simple que CPI(α)—est empiriquement proche de CPI(α), bien qu'étant en moyenne légèrement moins bon. CPI+, CPI(α) et PSDP$_\infty$ ont une performance moyenne similaire, mais la variabilité de PSDP$_\infty$ est significativement moindre. PSDP$_\infty$ apparaît ainsi comme donnant les meilleurs résultats empiriques. NSPI(m) constitue effectivement un pont entre API et PSDP$_\infty$. En augmentant m, le comportement de NSPI(m) se rapproche progressivement de celui de PSDP$_\infty$. Avec $m = 30$, NSPI(m) est globalement meilleur que API(α), CPI+, et CPI(α), et est proche de PSDP$_\infty$. L'ensemble de ces observations est plutôt stable par rapport aux paramètres n_S et n_A. De manière intéressante, les différences entre les algorithmes tendent à se dissiper lorsque la dynamique du problème est de plus en plus stochastique (lorsque le facteur de branchement augmente). Ceci est conforme à notre analyse basée sur les constantes de concentrabilité : celles-ci sont toutes finies lorsque la dynamique est fortement "mélangeante", et leur différences relatives sont les plus grandes pour des dynamiques déterministes.

5 Discussion, Conclusions et Perspectives

Dans cet article, nous avons considéré plusieurs variations d'itérations sur les politiques pour les problèmes à horizon infini : API, CPI, NSPI(m), API(α) et PSDP$_\infty$ [6]. Nous avons en particulier expliqué

5. Nous avons implémenté un mécanisme simpliste de recherche linéaire, qui considère l'ensemble des pas d'apprentissage $2^i\alpha$ où α est le pas minimal déterminé par CPI pour garantir une amélioration (cf. Kakade and Langford (2002)).

6. Nous rappelons qu'à notre connaissance, PSDP$_\infty$ (l'utilisation de PSDP dans un cas à horizon infini) n'est pas documentée dans la littérature.

le fait que l'algorithme récemment proposé NSPI(m) généralise API (obtenu lorsque $m = 1$) et PSDP$_\infty$ (lorsque $m = \infty$). La Figure 1 synthétise les garanties de performance de ces algorithmes. La plupart des bornes qui y sont décrites sont à notre connaissance nouvelles.

L'un des premiers messages important de notre travail est que ce qui est habituellement caché dans les constantes de concentrabilité compte. Les constantes impliquées dans les bornes d'API, CPI, PSDP$_\infty$ et pour les principaux termes (à gauche) de NSPI(m) peuvent être ordonnés de la pire à la meilleure comme suit : $C^{(2,1,0)}, C^{(1,0)}, C^{(1)}_{\pi_*}, C_{\pi_*}$. Une hiérarchie détaillée de toutes ces constantes est fournie à la Figure 1. A notre connaissance, c'est la première fois qu'une comparaison détaillée et complète des bornes de performance d'algorithmes de programmation dynamique est effectuée, et notre hiérarchie des constantes a des implications qui vont au-delà des schémas de type itération sur les politiques que nous avons considérés ici. En fait, plusieurs autres algorithms de programmation dynamique, notamment AVI (Munos, 2007), λPI (Scherrer, 2013), et AMPI (Scherrer *et al.*, 2012), ont des garanties qui sont exprimées en fonction de la pire constante $C^{(2,1,0)}$, ce qui suggère qu'ils ne devraient pas être compétitifs avec ceux que nous avons décrits ici.

D'un point de vue purement technique, plusieurs de nos bornes sont dérivées par paires ; ceci est dû au fait que nous utilisons une nouvelle technique de preuve. Celle-ci a permis de dériver une nouvelle borne de performance pour API, qui est meilleure que l'existant car elle implique la constance $C^{(1,0)}$ au lieu de $C^{(2,1,0)}$. Cela nous a également permis de dériver de nouvelles bornes pour CPI (et sa variation algorithmique CPI(α)) qui est moins bonne en termes de garantie mais meilleure du point de vue de la complexité en temps ($\tilde{O}(\frac{1}{\epsilon})$ au lieu de $O(\frac{1}{\epsilon^2})$). Nous pensons que cette nouvelle technique pourrait être utile dans de futures études sur la résolution approchée de PDMs.

Résumons les principaux enseignement de notre étude. 1) Les garanties pour CPI peuvent être arbitrairement meilleures que celles d'API/API(α), parce qu'elle sont exprimées en fonction de la meilleure constante de concentrabilité C_{π_*}, mais ceci se fait au prix d'une augmentation relative—exponentielle en $\frac{1}{\epsilon}$—du nombre d'itérations. 2) PSDP$_\infty$ combine les avantages d'API et CPI : sa garantie de performance est similaire à celle de CPI, mais elle est obtenue en un nombre d'itérations identique à celui d'API. 3) Contrairement à API qui recquiert une mémoire constante, la mémoire requise par CPI et PSDP$_\infty$ est proportionnelle au nombre d'itérations, ce qui peut être problématique lorsque le facteur d'actualisation γ est proche de 1, ou lorsque l'erreur d'approximation ϵ est proche de 0 ; nous avons montré que l'algorithme NSPI(m) permet de faire un compromis entre la qualité de performance de PSDP$_\infty$ et la mémoire d'API.

La limite principale de notre analyse réside en l'hypothèse, considérée tout au long du papier, que tous les algorithmes disposent d'un operateur approximativement gourmand de qualité ϵ. Ceci n'est en général par réaliste, car on ne peut pas contrôler directement le niveau de qualité ϵ. De plus, la comparaison de tous les algorithmes par rapport à cette quantité est limitée, car les problèmes d'optimisation sous-jacents sont de complexités variées : par exemple, les méthodes comme CPI cherchent dans un espace de politiques stochastiques, tandis qu'API se déplace dans un espace de politiques déterministes. Approfondir notre étude en explicitant ce qui est caché dans ce terme ϵ—comme nous l'avons fait ici pour les constantes de concentrabilité—constitue une perspective naturelle.

Dernier point, et non le moindre, du côté pratique, nous avons effectué des simulations numériques qui confirment notre analyse que des algorithmes avec de meilleures constantes de concentrabilité doivent être préférés. Sur plus de 800 PDMs Garnet avec diverses caractéristiques, CPI(α), CPI+ (CPI avec une recherche linéaire simpliste), PSDP$_\infty$ et NSPI(m) ont des performances qui surpassent significativement celles du schéma standard API. CPI+, CPI(α) et PSDP$_\infty$ ont des performances moyennes similaires, mais PSDP$_\infty$ est bien moins variable, ce qui fait de lui globalement le meilleur algorithme. Finalement, NSPI(m) permet de faire un pont entre API et PSDP$_\infty$, et permet d'atteindre des performances similaires à celle de PSDP$_\infty$ tout en contrôlant la mémoire. Considérer d'autres instances de ces schémas algorithmiques, analyser leurs performances et les tester empiriquement sur des problèmes de plus grande taille constitue l'une des perspectives les plus naturelles de ce travail.

A Preuves des bornes de performance (Table 1)

PSDP$_\infty$: Pour tout k, nous avons

$$\begin{aligned} v_{\pi_*} - v_{\sigma_k} &= T_{\pi_*} v_{\pi_*} - T_{\pi_*} v_{\sigma_{k-1}} + T_{\pi_*} v_{\sigma_{k-1}} - T_{\pi_k} v_{\sigma_{k-1}} \\ &\leq \gamma P_{\pi_*}(v_\pi - v_{\sigma_{k-1}}) + e_k \end{aligned}$$

où nous avons défini $e_k = \max_{\pi'} T_{\pi'} v_{\sigma_{k-1}} - T_{\pi_k} v_{\sigma_{k-1}}$. Comme P_π est positive, nous déduisons par induction :

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

En multipliant des deux côtés par μ, utilisant la définition des coefficients c_{π_*} et le fait que $\nu_j e_j \leq \epsilon_j \leq \epsilon$, nous obtenons :

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

La borne en fonction de $C^{(1)}_{\pi_*}$ est obtenue en utilisant le fait que $v_{\sigma_{k\cdots}} \geq v_{\sigma_k} - \gamma^k V_{\max}$ et en prenant $k \geq \left\lceil \frac{\log \frac{2V_{\max}}{\epsilon}}{1-\gamma} \right\rceil$. Repartant de l'Equation (7) et utilisant la définition de C_{π_*} (en particulier le fait que pour tout i, $\mu(\gamma P_{\pi_*})^i \leq \frac{1}{1-\gamma} d_{\pi^*,\mu} \leq \frac{C_{\pi_*}}{1-\gamma}\nu$) et le fait que $\nu e_j \leq \epsilon_j$, nous obtenons :

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

et la deuxième borne est obtenue en utilisant le fait que $v_{\sigma_{k\cdots}} \geq v_{\sigma_k} - \gamma^k V_{\max}$, $\sum_{i=1}^k \epsilon_i \leq k\epsilon$, et en considérant le nombre d'itérations $k = \left\lceil \frac{\log \frac{2V_{\max}}{\epsilon}}{1-\gamma} \right\rceil$.

API/NSPI(m) : API est identique à NSPI(1), et ses bornes sont des cas particuliers des deux premières bornes de NSPI(m), donc nous considérons seulement NSPI(m). En suivant la technique de preuve de Scherrer and Lesner (2012), notant $\Gamma_{k,m} = (\gamma P_{\pi_k})(\gamma P_{\pi_{k-1}}) \cdots (\gamma P_{\pi_{k-m+1}})$ et $e_{k+1} = \max_{\pi'} T_{\pi'} v_{\pi_{k,m}} - T_{\pi_{k+1}} v_{\pi_{k,m}}$, on peut montrer que :

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

En multipliant les deux côtés par μ (et observant que $e_k \geq 0$) et le fait que $\nu e_j \leq \epsilon_j \leq \epsilon$, nous obtenons

$$\mu(v_{\pi_*} - v_{\pi_k}) \leq \sum_{i=0}^{k-1} \mu(\gamma P_{\pi_*})^i (I - \Gamma_{k-i,m})^{-1} e_{k-i} + \gamma^k V_{\max} \qquad \text{:eq4}$$

$$\leq \sum_{i=0}^{k-1} \left(\sum_{j=0}^{\infty} \gamma^{i+jm} c(i+jm) \epsilon_{k-i} \right) + \gamma^k V_{\max} \qquad \text{:eq3}$$

$$\leq \sum_{i=0}^{k-1} \sum_{j=0}^{\infty} \gamma^{i+jm} c(i+jm) \epsilon + \gamma^k V_{\max}, \qquad \text{:eq5}$$

ce qui prouve la première borne en prenant $k \geq \left\lceil \frac{\log \frac{2V_{\max}}{\epsilon}}{1-\gamma} \right\rceil$. Repartant de l'Equation (A) et faisant l'hypothèse (pour simplifier l'écriture) que $\epsilon_{-k} = 0$ pour tout $k \geq 0$, nous obtenons

$$\begin{aligned}
\mu(v_{\pi_*} - v_{\pi_k}) - \gamma^k V_{\max} &\leq \sum_{l=0}^{\lceil \frac{k-1}{m} \rceil} \sum_{h=0}^{m-1} \sum_{j=0}^{\infty} \gamma^{h+(l+j)m} c(h+(l+j)m) \epsilon_{k-h-lm} \\
&\leq \sum_{l=0}^{\lceil \frac{k-1}{m} \rceil} \sum_{h=0}^{m-1} \sum_{j=l}^{\infty} \gamma^{h+jm} c(h+jm) \max_{k-(l+1)m+1 \leq p \leq k-lm} \epsilon_p \\
&\leq \sum_{l=0}^{\lceil \frac{k-1}{m} \rceil} \sum_{h=0}^{m-1} \sum_{j=0}^{\infty} \gamma^{h+jm} c(h+jm) \max_{k-(l+1)m+1 \leq p \leq k-lm} \epsilon_p \\
&= \left(\sum_{h=0}^{m-1} \sum_{j=0}^{\infty} \gamma^{h+jm} c(h+jm) \right) \sum_{l=0}^{\lceil \frac{k-1}{m} \rceil} \max_{l-(l+1)m+1 \leq p \leq k-lm} \epsilon_p \\
&\leq \left(\sum_{i=0}^{\infty} \gamma^i c(i) \right) \left\lceil \frac{k-1}{m} \right\rceil \epsilon, \qquad \text{:eq6}
\end{aligned}$$

ce qui prouve la deuxième borne en prenant $k = \left\lceil \frac{\log \frac{2V_{\max}}{\epsilon}}{1-\gamma} \right\rceil$. Finalement, repartant de l'Equation (A), et utilisant le fait que $(I - \Gamma_{k-i,m})^{-1} = I + \Gamma_{k-i,m}(I - \Gamma_{k-i,m})^{-1}$, on peut voir que

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

Le premier terme de la partie droite de l'inégalité peut être borné exactement comme dans le cas de PSDP$_\infty$. Pour le deuxième terme, nous avons

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

et on suit la même démarche que précédemment (de l'Equation (A) aux Equations (A) et (A)) pour conclure.

CPI, CPI(α), API(α) : Les pas conservatifs peuvent être analysés via une généralisation (un peu pénible) de l'analyse d'API par Munos (2003). La preuve est reportée à la partie "Compléments".

B Preuves des relations d'ordre entre les constantes de concentrabilité (Figure 1)

Nous donnons ici les détails qui permettent de déduire les relations entre constantes de concentrabilité.

$C_{\pi_*} \to C^{(1)}_{\pi_*}$ **:** (i) Nous avons $C_{\pi_*} \leq C^{(1)}_{\pi_*}$ puisque :

$$\begin{aligned}
d_{\pi_*,\mu} &= (1-\gamma)\mu(I - \gamma P_{\pi_*})^{-1} = (1-\gamma) \sum_{i=0}^{\infty} \gamma^i \mu (P_{\pi_*})^i \\
&\leq (1-\gamma) \sum_{i=0}^{\infty} \gamma^i c_{\pi_*}(i) \nu = C^{(1)}_{\pi_*} \nu
\end{aligned}$$

et le fait que C_{π_*} est la plus petite constante vérifiant cette inégalité. (ii) Nous pouvons avoir $C_{\pi_*} < \infty$ et $C^{(1)}_{\pi_*} = \infty$ en concevant un PDM sur $\mathbb{N}$ où π_* induit une transition déterministe de l'état i vers l'état $i+1$.

$\underline{C^{(1)}_{\pi_*} \to C^{(1,0)}}$ **:** (i) Nous avons $C^{(1)}_{\pi_*} \leq C^{(1,0)}$ parce que pour tout i, $c_{\pi_*}(i) \leq c(i)$. (ii) Il est facile d'obtenir $C^{(1)}_{\pi_*} < \infty$ et $C^{(1,0)} = \infty$ en utilisant deux politiques distinctes.

$\underline{C^{(1,0)} \to C^{(2,m,0)}}$ **and** $\underline{C^{(1,m)} \to C^{(2,m,m)}}$ **:** (i) $C^{(1,m)} \leq \frac{1}{1-\gamma^m} C^{(2,m,m)}$ est vrai car

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

(ii) On peut avoir $C^{(1,m)} < \infty$ et $C^{(2,m,m)} = \infty$ lorsque $c(i) = \Theta(\frac{1}{i^2\gamma^i})$, car le terme générique de $C^{(1,m)}$ est $\Theta(\frac{1}{i^2})$ (la série converge) tandis que celui de $C^{(2,m,m)}$ est $\Theta(\frac{1}{i})$ (la série diverge). Le raisonnement est similaire pour l'autre relation.

$\underline{C^{(1,m)} \to C^{(1,0)}}$ **and** $\underline{C^{(2,m,m)} \to C^{(2,m,0)}}$ **:** Nous faisons ici l'hypothèse que $m < \infty$. (i) Nous avons $C^{(1,m)} \leq \frac{1}{\gamma^m} C^{(1,0)}$ et $C^{(2,m,m)} \leq \frac{1}{\gamma^m} C^{(2,m,0)}$. (ii) Il suffit d'avoir $c(j) = \infty$ pour un $j < m$ pour avoir $C^{(2,m,0)} = \infty$ tandis que $C^{(2,m,m)} < \infty$, ou pour avoir $C^{(1,0)} = \infty$ tandis que $C^{(1,m)} < \infty$.

$\underline{C^{(2,1,0)} \leftrightarrow C^{(2,m,0)}}$ **:** (i) Nous avons clairement $C^{(2,m,0)} \leq \frac{1-\gamma^m}{1-\gamma} C^{(2,1,0)}$. (ii) $C^{(2,m,0)}$ peut être réécrit comme suit :

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

Alors, en utilisant le fait que $1 + \lfloor \frac{i}{m} \rfloor \geq \max\left(1, \frac{i}{m}\right)$, nous avons

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

Ainsi, quand m est fini, $C^{(2,m,0)} < \infty \Rightarrow C^{(2,1,0)} < \infty$.

Références

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

Bagnell, J., Kakade, S., Ng, A., and Schneider, J. (2003). Policy search by dynamic programming. In *Neural Information Processing Systems*, volume 16. MIT Press.

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

Kakade, S. (2003). *On the Sample Complexity of Reinforcement Learning*. Ph.D. thesis, University College London.

Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *ICML*, pages 267–274.

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

Lagoudakis, M. and Parr, R. (2003b). Reinforcement Learning as Classification : Leveraging Modern Classifiers. In *Proceedings of ICML*, pages 424–431.

Lazaric, A., Ghavamzadeh, M., and Munos, R. (2010). Analysis of a Classification-based Policy Iteration Algorithm. In *Proceedings of ICML*, pages 607–614.

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

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

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

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

Scherrer, B. and Lesner, B. (2012). On the Use of Non-Stationary Policies for Stationary Infinite-Horizon Markov Decision Processes. In *NIPS 2012 - Neural Information Processing Systems*, South Lake Tahoe, United States.

Scherrer, B., Gabillon, V., Ghavamzadeh, M., and Geist, M. (2012). Approximate Modified Policy Iteration. Research report, INRIA.

C Preuves pour CPI, CPI(α), API(α)

Nous commençons par montrer le résultat suivant :

Theorem 1. *A chaque itération $k < k^*$ de CPI (Equation (3)), l'espérance de la perte satisfait :*

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

Démonstration. En utilisant le fait que $T_{\pi_{k+1}} v_{\pi_k} = (1-\alpha_{k+1}) v_{\pi_k} + \alpha_{k+1} T_{\pi_{k+1}} v_{\pi_k}$ et la notation $e_{k+1} = \max_{\pi'} T_{\pi'} v_{\pi_k} - T_{\pi'_{k+1}} v_{\pi_k}$, nous avons :

$$\begin{aligned}
v_{\pi_*} - v_{\pi_{k+1}} &= v_{\pi_*} - T_{\pi_{k+1}} v_{\pi_k} + T_{\pi_{k+1}} v_{\pi_k} - T_{\pi_{k+1}} v_{\pi_{k+1}} \\
&= v_{\pi_*} - (1-\alpha_{k+1}) v_{\pi_k} - \alpha_{k+1} T_{\pi'_{k+1}} v_{\pi_k} + \gamma P_{\pi_{k+1}} (v_{\pi_k} - v_{\pi_{k+1}}) \\
&= (1-\alpha_{k+1})(v_{\pi_*} - v_{\pi_k}) + \alpha_{k+1}(T_{\pi_*} v_{\pi_*} - T_{\pi_*} v_{\pi_k}) + \alpha_{k+1}(T_{\pi_*} v_{\pi_k} - T_{\pi'_{k+1}} v_{\pi_k}) + \gamma P_{\pi_{k+1}} (v_{\pi_k} - v_{\pi_{k+1}}) \\
&\leq \left[(1-\alpha_{k+1}) I + \alpha_{k+1} \gamma P_{\pi_*}\right](v_\pi - v_{\pi_k}) + \alpha_{k+1} e_{k+1} + \gamma P_{\pi_{k+1}} (v_{\pi_k} - v_{\pi_{k+1}}). \qquad \text{:eq0}
\end{aligned}$$

En utilisant les faits que $v_{\pi_{k+1}} = (I - \gamma P_{\pi_{k+1}})^{-1} r$, et que $(I - \gamma P_{\pi_{k+1}})^{-1}$ est positive, on peut voir que

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

En remettant ceci dans l'Equation (C), on obtient :

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

Définissons la matrice $Q_k = [(1-\alpha_k) I + \alpha_k \gamma P_{\pi_*}]$, l'ensemble $\mathcal{N}_{i,k} = \{j; k-i+1 \leq j \leq k\}$ (cet ensemble contient exactement i éléments), la matrice $R_{i,k} = \prod_{j \in \mathcal{N}_{i,k}} Q_j$, et les coefficients $\beta_k = 1 - \alpha_k(1-\gamma)$ et $\delta_k = \prod_{i=1}^{k} \beta_k$. En utilisant plusieurs fois le fait que la matrice Q_k est positive, nous obtenons par induction

$$v_{\pi_*} - v_{\pi_k} \leq \sum_{i=0}^{k-1} R_{i,k} \alpha_{k-i} (I - \gamma P_{\pi_{k-i}})^{-1} e_{k-i} + \delta_k V_{\max}. \qquad \text{:eq0}$$

Soit $\mathcal{P}_j(\mathcal{N}_{i,k})$ l'ensemble des parties de $\mathcal{N}_{i,k}$ de taille j. Munis de cette notation, nous avons

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

où pour tout sous-ensemble I de $\mathcal{N}_{i,k}$, nous avons écrit

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

Par conséquent, en multipliant Equation (C) par μ, en utilisant la définition des constantes $c(i)$, et le

fait que $\nu \leq (1-\gamma) d_{\nu,\pi_{k+1}}$, nous obtenons :

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

Maintenant, utilisant le fait que pour tout $x \in (0,1)$, $\log(1-x) \leq -x$, nous pouvons observer que

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

En conséquence, nous obtenons $\delta_k \leq e^{-(1-\gamma)\sum_{i=1}^{k} \alpha_i}$. □

Dans l'analyse de CPI, Kakade and Langford (2002) montrent que les pas d'apprentissage qui garantissent la borne de performance satisfont $\alpha_k \geq \frac{(1-\gamma)\epsilon}{12\gamma V_{\max}}$; le terme $e^{\left\{(1-\gamma)\sum_{i=1}^{k} \alpha_i\right\}}$ tend vers 0 exponentiellement vite, et nous obtenons le corollaire suivant qui montre que CPI a une garantie exprimée avec le coefficient coefficient $C^{(1,0)}$ d'API en un nombre d'itérations $O(\frac{\log \frac{1}{\epsilon}}{\epsilon})$.

Corollary 1. *La plus petite itération (aléatoire) $k^\dagger$ telle que $\frac{\log \frac{V_{\max}}{\epsilon}}{1-\gamma} \leq \sum_{i=1}^{k^\dagger} \alpha_i \leq \frac{\log \frac{V_{\max}}{\epsilon}}{1-\gamma} + 1$ est telle que $k^\dagger \leq \frac{12\gamma V_{\max} \log \frac{V_{\max}}{\epsilon}}{\epsilon(1-\gamma)^2}$ et la politique $\pi_{k^\dagger}$ satisfait :*

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

Comme la preuve est basée sur une généralisation de l'analyse d'API, et n'utilise aucune des propriétés spécifiques de CPI, le résultat que nous venons d'énoncer peut facilement être spécialisé au cas de CPI(α).

Corollary 2. *Supposons que nous exécutons CPI(α) avec $\alpha \in (0,1)$, c'est-à-dire CPI (Equation (3)) avec $\alpha_k = \alpha$ pour tout k.*

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

La borne ci-dessus pour CPI(α) implique le facteur $\frac{1}{(1-\gamma)^3}$. Un examen précis de la preuve montre que cette amplification est due au fait que l'opérateur gourmand approché utilise la distribution $d_{\pi_k,\nu} \geq (1-\gamma)\nu$ au lieu de ν (pour API). En fait, en utilisant une preuve quasi similaire, on peut montrer le résultat suivant pour API(α).

Corollary 3. *Supposons que nous exécutons API(α) avec $\alpha \in (0,1)$.*

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

D Quelques détails supplémentaires sur les simulations

Problèmes et Approximations Dans nos expériences, un PDM Garnet est spécifié par 4 paramètres et est écrit $G(n_S, n_A, b, p)$: n_S est le nombre d'états, n_A est le nombre d'actions, b est le facteur de branchement qui indique combien d'états suivants peuvent être atteints pour chaque paire état-action courante (b états sont choisis uniformément au hasard et les probabilités de transition sont obtenues en échantillonnant uniformément $b-1$ point de coupes entre 0 et 1) et p le nombre de fonction caractéristiques (pour l'approximation linéaire). La récompense dépend seulement de l'état : pour un Garnet généré aléatoirement, la récompense pour chaque état est échantillonnée uniformément entre 0 et 1. Les fonctions caractéristiques sont choisies aléatoirement : Φ est une matrice de tialle $n_S \times p$ dont chaque élément est tiré aléatoirement uniformément entre 0 et 1. Le facteur d'actualisation γ est fixé à 0.99 dans toutes les expériences.

Tous les algorithmes que nous avons décrits dans le papier ont besoin d'accéder à un opérateur gourmand approché $\mathcal{G}_\epsilon(\rho, v)$ pour une distribution $\rho = \nu$ ou $\rho = d_{\pi,\nu}$. Pour implémenter cet opérateur, nous calculons un estimé bruité de la valeur v avec un bruit uniforme $u(\iota)$ d'amplitude ι, puis projetons cet estimé sur l'espace linéaire Φ par rapport à la projection orthogonale induite par la norme quadratique pondérée par ρ—projection que nous notons $\Pi_{\Phi,\rho}$—, puis nous application l'opérateur gourmand (exact) à ce projeté. Formellement, un appel de l'opérateur gourmand approché $\mathcal{G}_\epsilon(\rho, v)$ revient à calculer $\mathcal{G}\Pi_{\Phi,\rho}(v + u(\iota))$.

Simulations Nous avons lancé une série d'expériences, dans lesquelles les perturbations (bruit, espace linéaire) sont calibrées de sorte à ce que les algorithmes soient significativement (mais pas démesurément) perturbés. Après quelques essais et erreurs, nous avons considérés les réglages suivants. Nous utilisons des problèmes Garnet $G(n_S, n_A, b, p)$ avec $n_S \in \{50, 100, 200\}$, $n_A \in \{2, 5, 10\}$, $b \in \{1, 2, 10\}$ ($b = 1$ correspond à des problèmes déterministes), un espace linéaire d'approximation de taille $p = \frac{n_S}{10}$, et un bruit de niveau $\iota = 0.1$ (10%).

En complément de la Figure 2 qui montrent des statistiques globales sur l'ensemble des PDMs générés, les Figures 3, 4 et 5 montrent des statistiques qui sont conditionnées aux valeurs possibles de n_S, n_A et b, ce qui donne une idée de l'influence de ces paramètres.

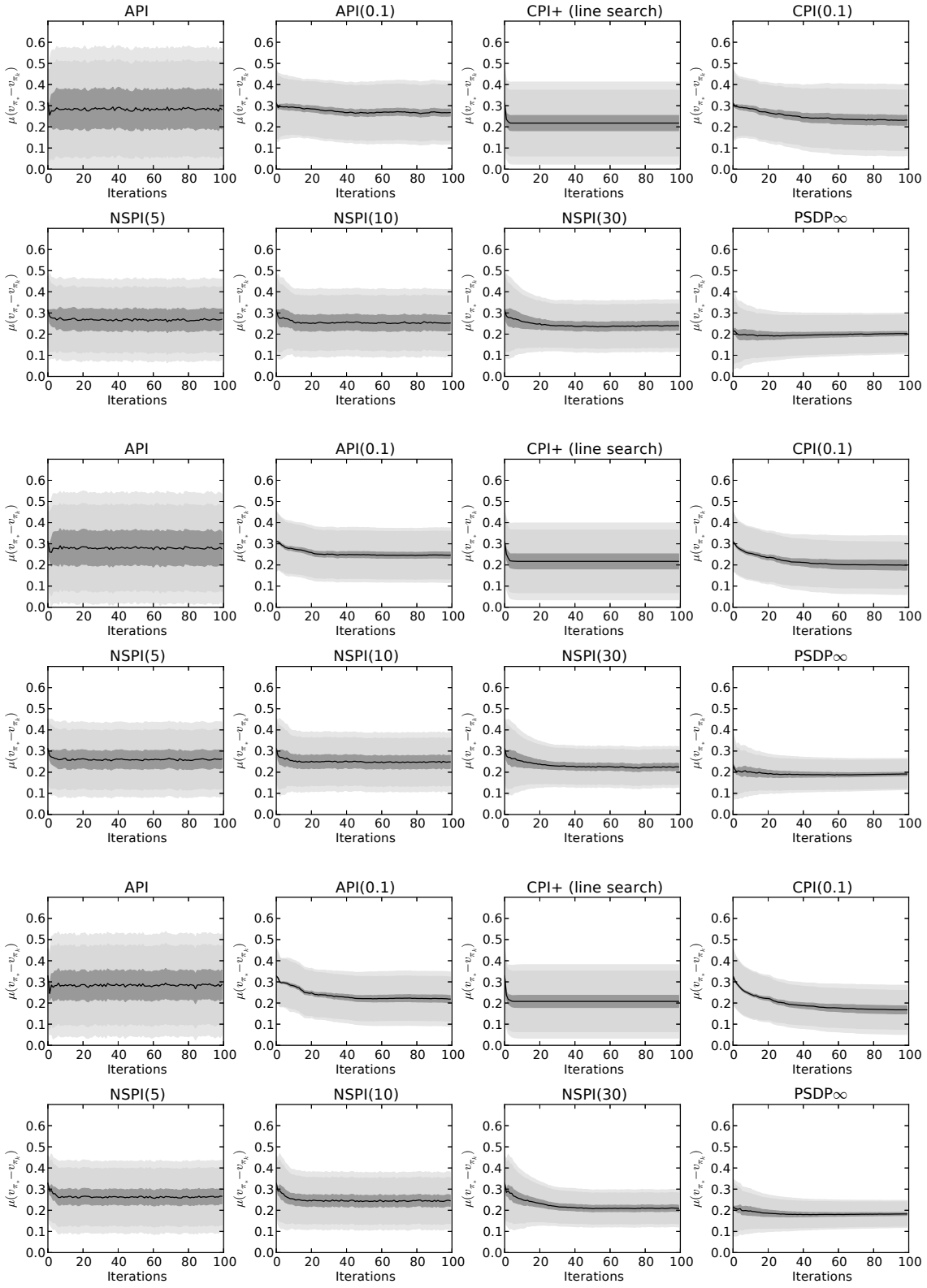

FIGURE 3 – Statistics conditioned on the number of states. Top : $n_S = 50$. Middle : $n_S = 100$. Bottom $n_S = 200$.

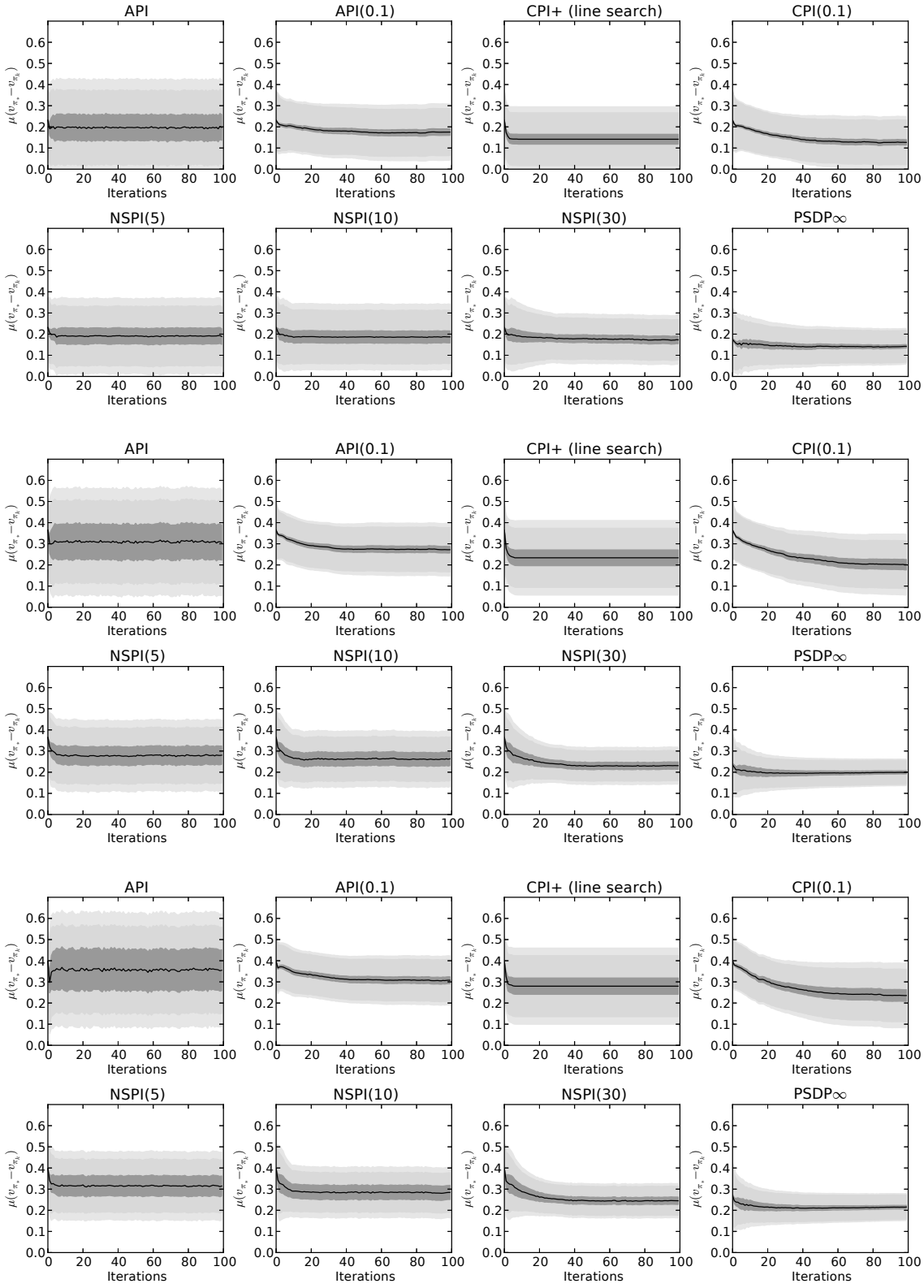

FIGURE 4 – Statistics conditioned on the number of actions. Top : $n_A = 2$. Middle : $n_A = 5$. Bottom $n_a = 10$.

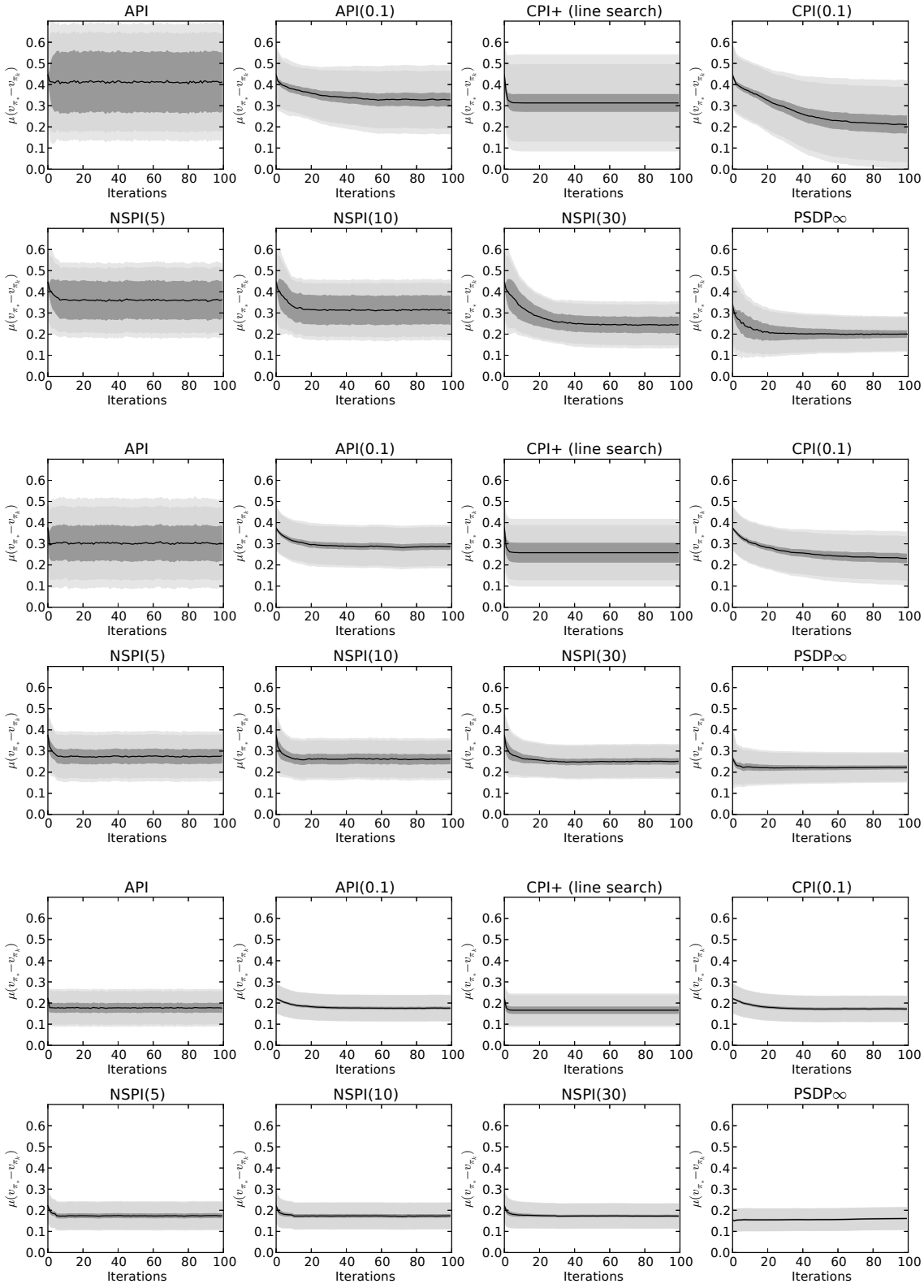

FIGURE 5 – Statistics conditioned on the branching factor. Top : $b = 1$ (deterministic). Middle : $b = 2$. Bottom $b = 10$.