Université des Sciences et des Technologies de Lille
École Doctorale Sciences Pour l'Ingénieur

Thèse de Doctorat

préparée au sein du lifl, UMR 8022 Lille 1/CNRS
et du centre de recherche Inria Lille - Nord Europe

Spécialité : **Informatique**

présentée par
Victor GABILLON

ALGORITHMES BUDGÉTISÉS D'ITÉRATION SUR LES POLITIQUES OBTENUES PAR CLASSIFICATION

sous la direction de **Mohammad GHAVAMZADEH**

Soutenue publiquement à **Villeneuve d'Ascq**, le **12 juin 2014** devant le jury composé de :

M. Peter **AUER**	Université de Leoben	Rapporteur
M. Olivier **CAPPÉ**	Télécom ParisTech	Examinateur
M. Mohammad **GHAVAMZADEH**	Inria Lille & Adobe	Directeur
M. Shie **MANNOR**	Technion	Rapporteur
M. Philippe **PREUX**	Université Lille 3	Directeur
M. Csaba **SZEPESVÁRI**	Université de l'Alberta	Examinateur

Résumé court en français : Cette thèse étudie une classe d'algorithmes d'apprentissage par renforcement (RL), appelée « itération sur les politiques obtenues par classification » (CBPI). Contrairement aux méthodes standards de RL, CBPI n'utilise pas de représentation explicite de la fonction valeur. CBPI réalise des déroulés (des trajectoires) et estime la fonction action-valeur de la politique courante pour un nombre limité d'états et d'actions. En utilisant un ensemble d'apprentissage construit à partir de ces estimations, la politique gloutonne est apprise comme le produit d'un classificateur. La politique ainsi produite à chaque itération de l'algorithme, n'est plus définie par une fonction valeur (approximée), mais par un classificateur. Dans cette thèse, nous proposons de nouveaux algorithmes qui améliorent les performances des méthodes CBPI existantes, spécialement lorsque le nombre d'interactions avec l'environnement est limité. Nos améliorations se portent sur les deux limitations de CBPI suivantes : 1) les déroulés utilisés pour estimer les fonctions action-valeur doivent être tronqués et leur nombre est limité, créant un compromis entre le biais et la variance dans ces estimations, et 2) les déroulés sont répartis de manière uniforme entre les états déroulés et les actions disponibles, alors qu'une stratégie plus évoluée pourrait garantir un ensemble d'apprentissage plus précis. Nous proposons des algorithmes CBPI qui répondent à ces limitations, respectivement : 1) en utilisant une approximation de la fonction valeur pour améliorer la précision (en équilibrant biais et variance) des estimations, et 2) en échantillonnant de manière adaptative les déroulés parmi les paires d'état-action.

English title: Budgeted Classification-based Policy Iteration

Short english abstract: This dissertation is motivated by the study of a class of reinforcement learning (RL) algorithms, called classification-based policy iteration (CBPI). Contrary to the standard RL methods, CBPI do not use an explicit representation for value function. Instead, they use rollouts and estimate the action-value function of the current policy at a collection of states. Using a training set built from these rollout estimates, the greedy policy is learned as the output of a classifier. Thus, the policy generated at each iteration of the algorithm, is no longer defined by a (approximated) value function, but instead by a classifier. In this thesis, we propose new algorithms that improve the performance of the existing CBPI methods, especially when they have a fixed budget of interaction with the environment. Our improvements are based on the following two shortcomings of the existing CBPI algorithms: 1) The rollouts that are used to estimate the action-value functions should be truncated and their number is limited, and thus, we have to deal with bias-variance tradeoff in estimating the rollouts, and 2) The rollouts are allocated uniformly over the states in the rollout set and the available actions, while a smarter allocation strategy could guarantee a more accurate training set for the classifier. We propose CBPI algorithms that address these issues, respectively, by: 1) the use of a value function approximation to improve the accuracy (balancing the bias and variance) of the rollout estimates, and 2) adaptively sampling the rollouts over the state-action pairs.

Mots clés: apprentissage automatique, prise de décision (statistique), algorithmes, intelligence artificielle, échantillonnage adaptatif (statistique), jeux de bandits, apprentissage par renforcement.

Key words: machine learning, decision-making, algorithm, artificial intelligence, adaptive sampling, bandit games, reinforcement learning.

Acknowledgement & Remerciements

Merci à tous pour m'avoir aidé à arriver au bout de cette thèse. Une grande partie de ce manuscrit repose sur votre travail et vos idées.

Mohammad, I have been really lucky to have you as my advisor. Thank you for being so constantly patient and supportive of me. All your advices and directions proved to lead for the best. I hope I have taken from you a bit of your vision and your rigour. Alessandro, working with you has been a real pleasure. I hope we will still continue to work together. Bruno, merci pour toutes les supers idées que tu as partagées avec moi, j'ai vraiment beaucoup appris grâce à toi, et merci pour ta gentillesse. Thanks to you Brano for the wonderful internship and all the support you gave me, I hope I managed to bring back with me a bit of your energy and inspiration. Thanks to my other co-authors, Matthieu, Boris, Sébastien, Brian, Zheng, and Muthu for all your help.

Thanks to Peter Auer, Shie Mannor, Olivier Cappé and Csaba Szepesvári for accepting to be part of my committee, especially to Peter and Shie who agreed to review this dissertation. It is a great honour for me.

Thanks to Christos Dimitrakakis who generously shared his code with me that I used as a base for my experiments. And huge thanks to you Sertan, without all your help, your advices and your ideas, a lot of the results in this dissertation would not exist.

Merci à toute l'équipe pour toute votre aide et votre sympathie : Hachem, Alexandra, Manuel, Odalric, Jean-François, Rémi C., Daniil, Nathan, Olivier N., Olivier P., Romaric, Marta, Adrien, Gergely, Gunnar, Balázs, Pierre, Michal, Prashanth, Tomáš, Amir, Raphael, Ronald, Sergio, Gabriel, Yasin, Matthew, Mohammad G. A., Sami, Lucian ainsi qu'Amélie et Sandrine. Et mersi Azal !!

Merci à tous ceux qui m'ont permis de me lancer dans la recherche. Merci à Jean-Pierre Delmas pour avoir guidé mes premiers pas ainsi qu'à Wojciech Pieczynski et Françoise Prêteux pour leurs conseils et leurs aides précieuses. Merci à Christophe Chailloux pour cette mémorable première expérience de recherche à l'étranger et pour tes conseils. Merci à Rémi Munos d'avoir suscité en moi la joie de faire du RL et de m'avoir permis de rejoindre son équipe. Merci à Jérémie Mary et Philippe Preux pour avoir encadré mon premier véritable travail de recherche, pour vos conseils et votre soutien qui a continué bien au delà du stage. Merci encore à vous deux et à Marc Tommasi, Dominique Gonzalez et Éric Wegrzynowski pour avoir guidé mes premiers pas dans l'enseignement.

Un gros bisous à toute ma famille pour leurs encouragements et leur soutien!

Merci à la plateforme d'essai Grid'5000, soutenue entre autres par l'INRIA, le CNRS, RENATER et plusieurs universités (voir https://www.grid5000.fr), qui m'a permis de réaliser la plupart des expériences rapportées dans cette dissertation sur leurs grilles d'ordinateurs.

Merci à l'INRIA pour le soutien financier et logistique ainsi qu'au ministère de la recherche et de l'enseignement supérieur et au conseil régional du Nord-Pas de Calais pour leur soutien financier.

Abstract.

This dissertation is motivated by the study of a relatively recent class of approximate dynamic programming (ADP) or reinforcement learning (RL) algorithms, called classification-based policy iteration (CBPI). CBPI is a class of ADP/RL algorithms that, contrary to the standard ADP/RL methods, do not use an explicit representation for value function. Instead, they use rollouts and estimate the action-value function of the current policy at a collection of states, called the rollout set. They then build a training set that consists of the states in the rollout set paired with their greedy actions (the action with the highest estimated value function). Using this training set, the greedy policy (with respect to the current policy) is learned as the output of a classifier. Thus, the policy generated at each iteration of the algorithm, is no longer defined by a (approximated) value function, but instead by a classifier. Given the structure of the CBPI algorithms, it is natural to think that they should perform better than their value function-based counterparts in problems in which the policies are simpler to represent, and thus to learn, than their corresponding value functions.

In this thesis, we propose new algorithms that improve the performance of the existing CBPI methods, especially when they have a fixed budget of interaction with the environment (or fixed number of question from the generative model of the problem). Our improvements are based on the following two shortcomings of the existing CBPI algorithms that play a crucial role in case of limited budget: **1)** The rollouts that are used to estimate the action-value functions should be truncated, and thus, we have to deal with bias-variance tradeoff in estimating the rollouts (large bias and small variance for short, and small bias and large variance for long rollouts), and **2)** The rollouts are allocated uniformly over the states in the rollout set and the available actions, while a smarter allocation strategy could guarantee a more accurate training set for the classifier, and as a result, a better performance for the overall algorithm. We propose CBPI algorithms that address these issues, respectively, by: **1)** the use of a value function approximation to improve the accuracy (balancing the bias and variance) of the rollout estimates, an idea similar to that of the actor-critic algorithms, and **2)** adaptively (rather than uniformly) sampling the rollouts over the state-action pairs.

To address the bias-variance tradeoff in the rollout estimates, our idea is to estimate each action-value function as a combination of a truncated rollout and a value function approximator, called the *critic*, that approximates the value at the state at which the rollout is truncated. If the quality of the critic is good, we can rely on it and shorten the rollouts, which in turn, reduces both the bias and variance of the estimates. The variance is reduced because the rollouts become shorter, and the bias is reduced as long as the critic provides a better estimate than truncation, i.e., replacing the value at the truncated state with zero. Using this idea, we present a new CBPI algorithm, called *direct policy iteration with a critic* (DPI-Critic), and provide its finite-sample performance analysis. Our analysis and empirical evaluations in two benchmark RL

problems allow us to characterize how our new approach can improve the overall performance of the the CBPI algorithms by tuning the length of the rollouts as a function of the quality of the critic. We further explore this idea and propose a CBPI algorithm that belongs to the class of Approximate Modified Policy Iteration algorithms (AMPI). Modified Policy Iteration (MPI) generalizes the well-known value and policy iteration algorithms. Despite its generality, MPI has not been studied and analyzed in its approximate form. We propose three AMPI algorithms, one belongs to the class of CBPI methods, and provide their finite-sample performance analysis, the first reported analysis for this class of ADP algorithms.

Another contribution of this thesis is on the applicability of the CBPI algorithms, and their potential advantage with respect to their value function-based counterparts. We apply our proposed AMPI-base CBPI algorithm, called *classification-based modified policy iteration* (CBMPI), to the game of Tetris. Tetris is a well-studied and challenging optimization problem at which ADP/RL algorithms have shown poor performance, while policy search techniques, mainly based on the cross entropy (CE) method, have produced controllers with excellent performance. We use the CBMPI algorithm and learn controllers that achieve better performance than those learned by CE, with considerably fewer number of samples. These are the best (to the best of our knowledge) controllers for the game of Tetris that have been reported in the literature.

Finally, we address the rollout allocation problem in CBPI by formulating it as the problem of identifying the best arm(s) (each arm corresponds to an available action) at each of the bandits (each bandit corresponds to a state in the rollout set) in a multi-bandit multi-armed setting. In addition to rollout allocation in CBPI, the problem of multi-bandit best arm(s) identification has application in a number of different domains such as clinical trials, network optimization, and brain computer interface. In order to solve this problem, we propose two algorithms, called *Gap-based Exploration* (GapE) and *Unified Gap-based Exploration* (UGapE), that focus on the arms with small gap, i.e., an arm whose mean is close to the mean of the best arm in the same bandit. For each algorithm, we prove an upper-bound on its probability of error. These are the first algorithms with complete analysis for the problem of multi-bandit best arm(s) identification. We evaluate the performance of our algorithms and compare them against other allocation strategies in a number of synthetic problems, in a real-word clinical problem, and in the CBPI's rollout allocation problem.

Résumé en français

Le problème de la prise de décisions séquentielles dans l'incertain est un problème de la vie courante. Nous le rencontrons lorsque nous nous rendons au travail depuis notre maison, lorsque nous jouons (au backgammon, au poker, ou au jeu de Tetris qui sera utilisé comme terrain d'expérience dans cette thèse), lors de nos recherches sur internet, ou si nous désirons optimiser les performances de notre entreprise, etc. Parmi ces problèmes d'intérêt et de prise de décisions séquentielles, nombreux sont ceux qui peuvent se formuler comme des problèmes d'apprentissage par renforcement (*reinforcement learning*, RL). En apprentissage par renforcement, un agent interagit avec un environnement dynamique, stochastique et qu'il ne connaît que partiellement. Son but est d'apprendre une stratégie, aussi appélée *politique*, qui optimise une certaine mesure de performance à long-terme (le nombre de lignes disparues pour le cas du jeu de Tetris)

L'apprentissage par renforcement a obtenu de nombreux succès dans des domaines variés comme pour le jeu de Backgammon (Tesauro, 1994), la gestion automatique d'une centrale d'appels (Marbach and Tsitsiklis, 1997), la commande de vol d'hélicoptère (Ng et al., 2004), l'envoi intelligent de catalogues (Simester et al., 2006), et la gestion de systèmes de dialogues parlés (Pietquin et al., 2011). Malgré ces succès, il reste des obstacles fondamentaux entravant l'application généralisée des méthodes d'apprentissage par renforcement aux problèmes complexes du monde réel. En effet, ces problèmes concrets se caractérisent souvent par les points limitant suivants :

- Le système (l'environnement) considéré est composé de large espaces d'actions et d'états, voire même d'espaces de taille infinie. Des techniques d'approximation sont alors requises afin de représenter les politiques et/ou les fonctions (action-)valeur (une fonction (action-)valeur est une fonction qui associe à un état du système un nombre réel qui mesure l'espérance de la performance de la politique lorsque cet état (et cette action) est pris comme état initial (et comme action initiale) du système)

- Les données d'apprentissage sont coûteuses. À la différence de l'apprentissage supervisé, pour lequel de vastes lots de données sont habituellement disponibles et immédiatement utilisables, en apprentissage par renforcement, les données d'apprentissage sont produits par l'agent lui-même lors de son interaction avec le système dynamique (que ce système soit réel ou simulé) et alors même que l'agent essaie de mieux le contrôler. Dans le cas de systèmes complexes, ceci peut se traduire par un coût prohibitif pour chaque nouvel échantillon d'apprentissage.

- La nécessité d'un apprentissage en-ligne. L'apprentissage hors-ligne est adéquat si l'environnement est stationnaire, ce qui n'est que rarement le cas.

- L'observation partielle. Dans de nombreux problèmes, l'état du système n'est pas complement ou directement observable par l'agent.

Dans cette thèse, nous nous concentrons sur les deux premières caractéristiques énoncées ci-dessus dont nous abordons les problématiques dans le contexte d'une classe d'algorithmes d'apprentissage par renforcement relativement nouvelle, appelée itération sur les politiques obtenues par classification (*classification-based policy iteration*, CBPI). CBPI est une variante de l'algorithme approximatif d'itération sur les politiques (*approximate policy iteration*, API), un algorithme de programmation dynamique (PD), qui remplace les étapes d'*évaluation de la politique* (approximant la fonction valeur de la politique courante sur la totalité de l'espace des états) et d'*amélioration de la politique* (produisant une nouvelle politique dont les performances sont supérieurs à la fonction valeur de la politique courante) par une étape d'apprentissage dans un espace de politique. Plus précisément, l'idée principale est de remplacer l'étape d'évaluation de la politique par des estimations *déroulées* (rollout estimates) de la fonction action-valeur pour un nombre fini d'états, appelé l'*ensemble déroulé* et pour chaque action possible ; et de formuler l'étape d'amélioration de la politique comme un problème de classification. L'ensemble d'apprentissage de ce problème de classification (dans sa forme la plus simple) est composé, en entrée, par les états de l'ensemble déroulé et, en sortie, par les actions ayant la fonction valeur estimée la plus grande pour chaque état.

Cette classe d'algorithmes de type API a été introduite par Lagoudakis and Parr (2003b) et Fern et al. (2004) (voir aussi la version journal Fern et al. 2006), puis définie plus précisément ainsi qu'analysée de manière complète par Lazaric et al. (2010a). Les résultats théoriques et empiriques obtenus par les algorithmes CBPI indiquent qu'ils constituent une alternative aux méthodes API standards se reposant sur des fonctions valeur (e.g., l'itération sur les politiques à l'aide des moindres carrés Lagoudakis and Parr 2003a), pouvant même se montrer supérieurs à ces derniers lorsque des politiques de qualité sont plus faciles à représenter, et par là même à apprendre que leurs fonctions valeur (e.g., au jeu de Tetris, comme nous le montrerons au Chapitre 4 de cette thèse).

Dans la première partie de cette thèse, aux Chapitres 3 et 4, nous proposons deux extensions des algorithmes CBPI existants qui améliorent la qualité des estimations déroulées des fonctions action-valeur, et par conséquent, augmentent les performances globales de l'algorithme. Pour chacun de ces algorithmes, une analyse des performances à échantillons finis est produite et une évaluation empirique de leurs performances comparées avec des algorithmes similaires est menée. Dans la deuxième partie de la thèse, Chapter 5, nous proposons des méthodes issues de la littérature sur les *bandits* afin de parfaire la stratégie de répartition des déroulés dans les algorithmes CBPI. L'objectif principal est d'allouer un budget limité de déroulés aux états de l'ensemble déroulé et aux actions disponibles de manière à obtenir l'ensemble d'apprentissage le plus précis pour le classificateur. Dans cette dissertation, le problème de répartition des déroulés pour CBPI est formulé comme une classe de problème de bandits, appelée d'*exploration pure*, ou plus spécifiquement, comme un problème de la classe, appelée *identifications des meilleurs bras en contexte multi-bandits*. De nouveaux algorithmes sont développés pour ce problème et soutenus par des garanties théoriques. L'application de nos algorithmes dépasse le problème de la répartition pour CBPI, car cette classe d'algorithmes de bandits est directement reliée au problème central qu'est l'allocation dynamique des ressources et qui trouve de nombreux champs d'applications allant du marketing et de la publicité aux études cliniques et aux réseaux de communications.

Motivation

Prenons le jeu de Tetris comme exemple du problème de la prise de décisions séquentielles dans l'incertain. Comme illustré à la Figure 1.1 visible en page 4, Tetris est un jeu qui se déroule sur un tableau rectangulaire composé à l'origine de 20 lignes et 10 colonnes, où des pièces composées de quatre briques et pouvant prendre sept formes différentes chutent depuis le haut du tableau les unes après les autres. Étant donné la configuration présente du tableau et la nouvelle pièce, qui définissent ensemble l'état actuel du jeu (du système), le joueur (l'agent) sélectionne une action en décidant à la fois de la position finale de la nouvelle pièce et de son orientation. Pour chaque ligne entièrement remplie, une récompense de 1 est reçue, les briques de cette ligne disparaissent et toutes les briques situées au dessus descendent d'une case. La transition entre les états du système est incertaine car la forme de la nouvelle pièce est tirée aléatoirement. L'objectif est de faire disparaître un maximum de lignes de briques avant la fin du jeu qui advient lorsqu'une pièce dépasse du tableau. Ce jeu a été souvent utilisé comme un problème d'optimisation de référence dans lequel le but est de concevoir un contrôleur (une politique) qui maximise la moyenne (sur de multiples parties) du nombre de lignes disparues (le score). Ce problème d'optimisation est connu pour nécessiter d'intense calculs. Il contient un nombre extrêmement grand de configurations possibles du tableau (environ $2^{200} \simeq 1.6 \times 10^{60}$), et même en considérant le cas où la séquence des pièces est connue à l'avance, trouver la stratégie qui maximise le score est un problème NP-difficile (Demaine et al., 2003).

Le jeu de Tetris peut être facilement formulé comme un processus de décision markovien (PDM). Dès lors, on peut tenter de le résoudre à l'aide d'algorithmes de programmation dynamique (PD) et d'apprentissage par renforcement (RL) (c'est à dire trouver un bon contrôleur pour ce jeu). Cependant, de par la taille immense de l'espace des états dans ce jeu, ces algorithmes ne peuvent être utilisés que dans leur forme approximative. La plus grande partie des algorithmes de programmation dynamique approximative et de RL se repose sur des fonctions valeur, c'est à dire, sur l'utilisation d'un espace de fonctions dans lequel les fonctions valeur sont approximées. La qualité de la solution produite par ces algorithmes dépend du choix de l'espace de fonctions (de sa capacité à approximer les fonctions valeur des politiques produites à chaque itération de l'algorithme) et du nombre d'échantillons utilisés pour calculer l'approximation (dans l'espace de fonction) de chaque fonction valeur. Des algorithmes d'ADP et de RL se reposant sur des fonctions valeurs ont été appliqués au jeu de Tetris (Tsitsiklis and Van Roy, 1996, Bertsekas and Ioffe, 1996, Farias and Van Roy, 2006, Scherrer, 2013), mais leurs performances restent inférieures par plusieurs ordres de grandeurs aux techniques de «l'état de l'art» qui, elles, cherchent directement dans un espace de politiques en apprenant les paramètres des politiques grâce à des outils d'optimisation, comme la méthode de l'entropie croisée (CE) (Szita and Lőrincz, 2006, Thiéry and Scherrer, 2009b). Ces résultats renforcent la conjecture que le jeu de Tetris est peut être un problème de RL où les politiques peuvent être plus aisément paramétrées, et donc apprises, que leurs fonctions valeur associées (par exemple si les politiques sont des fonctions régulières alors que leurs fonctions valeurs sont très bruitées et complexes). Ils font de Tetris un client naturel pour la classe relativement nouvelle des algorithmes d'ADP, appelée itération sur les politiques obtenues par classification (CBPI) (Lagoudakis and Parr, 2003b, Fern et al., 2004, 2006, Lazaric et al., 2010a).

Les algorithmes CBPI fonctionnent ainsi : à chaque itération, étant donné une politique courante : **1)** un *ensemble déroulé* est formé en échantillonnant (selon une distribution sur les états) un certain nombre d'états dans l'espace des états du problème ; **2)** pour chaque état dans l'ensemble déroulé et pour chaque action dans l'espace des actions du problème, un certain nombre de déroulés (de trajectoires) sont exécutés à partir desquels sont calculées des estimations déroulées des fonctions action-valeur pour ces paires d'état-action ; **3)** un ensemble d'apprentissage est construit à partir (dans sa forme la plus simple) des états de l'ensemble déroulé comme entrée et des actions pour lesquelles la fonction actions-valeurs estimée est la plus haute en sortie ; et finalement **4)** l'ensemble d'apprentissage est utilisé pour apprendre un classificateur, dont le produit est une estimation de la politique gloutonne (grossièrement parlant, une politique dont la performance est au moins aussi bonne que celle de la politique courante) par rapport à la politique courante. Ce classificateur peut être considéré comme l'espace des politiques où sont approximées les politiques gloutonnes.

Concernant la précision des estimations de la fonction action-valeur, il est important de noter que les estimations déroulées des fonctions action-valeur ne sont pas biaisées (si le déroulé est suffisamment long, c'est à dire constitué d'un grand nombre de pas), mais qu'elles pâtissent une grande variance (la variance augmente avec la longueur du déroulé). En découle une question importante : Si l'on fixe un budget pour le nombre total d'échantillons (de pas) que comprennent l'ensemble des déroulés, comment produire des estimations précises des fonctions action-valeur dans CBPI ? C'est à cette question que nous essayons de répondre dans la première partie de la thèse, aux Chapitres 3 et 4. Dans cette partie, nous proposons plusieurs méthodes ayant pour but de trouver un bon équilibre entre le biais et la variance des estimations et nous les appuyons avec une analyse théorique ainsi que des résultats empiriques.

Une autre question importante se pose : étant donné un ensemble déroulé, comment répartir un budget fixe de déroulés entre ces états et entre les actions de l'espace des actions afin d'obtenir un ensemble d'apprentissage précis pour le classificateur ? Il parait naturel d'imaginer que détecter l'action gloutonne (l'action pour laquelle la fonction action-valeur estimée est la plus grande) est plus difficile dans certains états que dans d'autres, et que, par conséquent, la répartition uniforme communément utilisée peut être inefficace. Cette question motive la seconde partie de cette dissertation (Chapitre 5). Nous abordons cette question en la formulant comme un problème de bandit à bras multiples, puis en développant des algorithmes de bandits associés à des garanties théoriques. Ainsi, nos algorithmes ne sont pas limités aux problèmes de répartition des déroulés dans CBPI et peuvent être utilisés dans le cadre plus général, et de plus large intérêt, qu'est la classe de problèmes appelée l'*allocation dynamique des ressources*.

Notre Approche

Comme expliqué ci-dessus, la contrainte sur le budget imposée dans CBPI force à tronquer les déroulés après un nombre fini de pas, m. Alors que la variance des estimations déroulées s'en trouve réduite, le biais ainsi introduit peut potentiellement affecter les performances globales de l'algorithme. Pour traiter ce compromis biais-variance, nous proposons tout d'abord (au Chapitre 3) d'utiliser une approximation de la fonction valeur, appelé *critique*, qui, associée aux résultats des déroulés, renvoie une estimation des fonctions action-valeur. Plus précisément, la critique renvoie une estimation de la fonction valeur pour l'état auquel le déroulé a été tronqué. L'algorithme qui implémente cette idée s'appelle *itération directe sur les politiques munie d'une critique* (DPI-Critique) par référence à l'algorithme de type CBPI appelé itération directe sur les politiques (*direct policy iteration*, DPI) (Lazaric et al., 2010a).

Cette idée est similaire aux principes des algorithmes *acteur-critique* qui font partie des premiers algorithmes développer en RL (Barto et al., 1983, Sutton, 1984). Si nous disposons d'une critique avec un espace de fonctions riche, il est raisonnable de raccourcir les déroulés et de donner plus d'importance à la critique, néanmoins, apprendre une critique dans un espace riche peut nécessiter un nombre conséquent d'échantillons. De manière similaire, pour des critiques avec un espace plus restreint, les déroulés doivent être allongés, ce qui augmente la part des échantillons réservée aux déroulés et restreint encore d'avantage la part réservée à la critique. Enfin, il faut ajouter à cela un autre compromis provoqué par la limitation du nombre total de pas dans les déroulés. Car alors, allonger les déroulés pour limiter le biais revient à diminuer le nombre d'états dans l'ensemble déroulé et ainsi à réduire la précision du classificateur. Notre analyse théorique et nos résultats expérimentaux pour cet algorithme montrent qu'il est possible, étant donné le budget total d'échantillons, de trouver un compromis entre la richesse de la critique, la longueur des déroulés et le nombre d'états déroulés. Et ceci dans le but d'obtenir la plus grande précision possible dans l'estimation des fonctions action-valeur pour le budget fixé.

L'idée de réaliser un déroulé de m pas puis de combiner son résultat avec la fonction valeur de l'état auquel il a été tronqué est similaire à l'application sur cette fonction valeur de l'opérateur de Bellman à m pas. Cette utilisation de l'opérateur de Bellman à m pas est la marque de fabrique de l'algorithme d'*itération modifiée sur les politiques* (*modified policy iteration*, MPI) (Puterman and Shin, 1978) qui généralise les méthodes classiques d'itération sur les valeurs et sur les politiques pour $m = 1$ et $m = \infty$, respectivement. Dans le Chapitre 4, nous abordons une nouvelle fois notre problématique en développant une version de MPI où les politiques sont obtenues par classification, appelé CBMPI, qui utilise un classificateur pour approximer les politiques en plus de l'utilisation (standard pour MPI) d'une critique. En plus de CBMPI, nous développons deux algorithmes MPI approximatifs (AMPI), et pour chacun des algorithmes nous fournissons leur analyse en échantillons finis (la première du genre pour tous ces algorithmes), et nous évaluons leurs performances et les comparons avec des méthodes similaires dans le Chapitre 4. L'utilisation de CBMPI nous a permis de trouver le meilleur contrôleur rapporté pour le jeu de Tetris. Dans nos expériences, nous montrons que CBMPI obtient (en moyenne) des performances similaires aux méthodes d'entropie croisée (qui forment «l'état de l'art» pour ce jeu) tout en utilisant considérablement moins d'échantillons.

Dans le seconde partie de la thèse, Chapitre 5, nous portons notre attention sur la deuxième question posée dans la Section 1, comment répartir les déroulés afin d'obtenir un ensemble d'apprentissage précis pour le classificateur. La précision de l'ensemble d'apprentissage dépend de notre capacité à identifier l'action gloutonne (l'action pour laquelle la fonction action-valeur est la plus élevée) pour les états de l'ensemble déroulé. Notons qu'à chaque réalisation d'un déroulé pour une paire état-action, un échantillon aléatoire tiré d'une distribution est observé, dont l'espérance est la valeur de la fonction action-valeur pour cette paire état-action. Par conséquent, nous pouvons naturellement considérer qu'il existe, pour chaque état de l'ensemble déroulé, un certain nombre de distributions inconnues (égal au nombre d'actions possibles dans cet état), et que le but est de les échantillonner de manière à détecter, le plus rapidement possible, celle dont l'espérance la plus élevée. Ce problème a été étudié dans le cadre des bandits à bras multiples sous le nom d'*identification du meilleur bras* (Maron and Moore, 1993, Bubeck et al., 2009), et plusieurs algorithmes efficaces ont été conçus pour le résoudre (Audibert et al., 2010). En se conformant à ce cadre, un état de l'ensemble déroulé devient un bandit ; une action disponible dans un état est un bras de ce bandit ; le tirage d'un bras revient à réaliser un déroulé et à recevoir un échantillon de la distribution dont l'espérance est la valeur de la fonction action-valeur pour cette paire état-action ; et l'objectif est de répartir le budget disponible (compté en nombre de déroulés, ou tirages) de manière à détecter le bras ayant l'espérance la plus haute avec grande probabilité. Cependant, l'important dans notre cas est d'identifier le plus précisément possible l'action gloutonne pour tous les états de l'ensemble déroulé et non pas seulement pour un seul d'entre eux. Il est donc nécessaire d'étendre les algorithmes de bandits existants pour l'identification du meilleur bras dans un contexte à bandits multiples. Nous montrons dans le Chapitre 5 que cette extension n'est pas évidente, ou plutôt que les solutions triviales mènent à des algorithmes peu efficaces. Par la suite, nous développons un premier algorithme pour l'identification des meilleurs bras en contexte multi-bandits pourvu de son analyse théorique et montrons ces performances dans plusieurs problèmes synthétiques ainsi que sur des données médicales. Le problème d'identification du meilleur bras, que ce soit pour un brandit simple ou pour de multiples bandits, n'est pas limité à la répartition des déroulés dans CBPI. Il est directement lié au problème important de l'*allocation dynamique des ressources* dont les nombreux champs d'applications vont du marketing et de la publicité aux études cliniques et aux réseaux de communications.

De plus cette version du problème possède d'autres applications potentielles telles que le problème clinique suivant. Soient M groupes de patients, pour lesquels un traitement doit être sélectionné parmi les K_p options associées à chaque groupe p. Un groupe peut correspondre à des patients ayant les mêmes marqueurs biologiques ou génétiques et les options sont des traitements pour une maladie donnée. L'objectif principal ici est de déterminer une règle qui recommande le meilleur traitement pour chaque groupe de patients. Ces règles sont généralement construites en utilisant des données de tests cliniques coûteux à réaliser. En conséquence, il est important de distribuer les tests intelligemment de manière à ce que la règle ainsi apprise donne de bons résultats. Comme il peut être plus difficile de trouver le meilleur traitement pour certains groupes que pour d'autres, la stratégie consistant à tester les patients par ordre d'arrivée peut ne pas aboutir à de bonnes performances. De la même manière, répartir les traitements uniformément aux différents groupes ne permettra sûrement pas de découvrir les meilleurs traitements chez certains des groupes. Ce problème peut être formuler comme une tentative d'*identification des meilleurs bras* parmi M bandits à bras multiples. Dans cette formulation, chaque groupe de patients est considéré comme un bandit à bras multiples, et chaque traitement comme un bras, tester un traitement sur un patient revient à un tirage de bras, et notre but est de recommander un bras pour chaque bandit après un nombre donné de tirages (budget). Les résultats peuvent être évalués par **1)** la moyenne sur les bandits de la valeur des bras recommandés, ou par **2)** la probabilité moyenne d'erreur sur les bandits (de ne pas sélectionner le bon bras), ou encore par **3)** la probabilité maximale d'erreur parmi les bandits.

Contributions

Les principales contributions de cette dissertation sont résumées ci-dessous.

Chapitre 3 : Introduction d'une Critique dans les Algorithmes CBPI

- Nous proposons un algorithme, appelé *itération directe sur les politiques munie d'une critique* (DPI-Critique), qui ajoute une composante issue de l'approximation des fonctions valeurs à l'estimation déroulée des fonctions action-valeur dans les algorithmes CBPI. Plus précisément, DPI-Critique est une extension d'un algorithme CBPI, appelé *itération directe sur les politiques* (DPI) proposé et analysé par Lazaric et al. (2010a).

- Nous analysons les performances de l'algorithme DPI-Critique lorsque la critique est basée sur les méthodes **1)** de différence temporelle basée sur les moindres carrés (LSTD) (Bradtke and Barto, 1996) et **2)** de minimisation du résidu de Bellman (BRM) (Baird, 1995). Dans chaque cas, nous montrons comment les erreurs produites par l'algorithme à chaque itération se propagent à travers les itérations et nous fournissons une analyse en échantillons finis de la performance après K itérations. Les résultats théoriques indiquent qu'à budget fixé, DPI-Critique peut obtenir de meilleurs performances que DPI, et l'itération sur les politiques basées sur les moindres carrés (LSPI) (Lagoudakis and Parr, 2003a). Ce succès dépend du réglage de plusieurs paramètres dont la longueur des déroulés et la qualité de la critique.

- Nous évaluons les performances de DPI-Critique et les comparons avec DPI et LSPI dans les problèmes de la *voiture tout terrain* et du *pendule inversé*. Les résultats empiriques renforcent notre analyse théorique et confirment que DPI-Critique peut mettre à la fois à profit les déroulés et la critique pour surpasser DPI et LSPI.

Chapitre 4 : Algorithme Approximatif d'Itération Modifiée sur les Politiques

- Nous proposons trois variantes d'algorithmes approximatifs MPI (AMPI), appelées AMPI-V, AMPI-Q, et CBMPI, correspondant à trois célèbres algorithmes de programmation dynamique approximative (ADP), respectivement : itération ajustée sur les valeurs (Munos and Szepesvári, 2008), Q-itération ajustée (Ernst et al., 2005, Antos et al., 2007), et itération sur les politiques obtenues par classification (CBPI). Il convient de signaler que l'implémentation d'AMPI où les politiques sont obtenues par classification donne une nouvelle vue de cette classe d'algorithmes, complémentaire avec celle proposée par l'algorithme DPI-Critique, proposé et analysé au Chapitre 3.

- Nous rapportons la première analyse de propagation d'erreur pour AMPI unifiant celle des algorithmes approximatifs d'itération sur les politiques et les valeurs. Nous fournissons aussi l'analyse en échantillon finis de ces trois algorithmes de type AMPI. Nos résultats indiquent que le paramètre libre de MPI permet de contrôler la répartition des erreurs (entre l'approximation des fonctions valeur et celle des politiques) dans la performance finale de l'algorithme CBPI.

- La performance de CBPI et le rôle du paramètre libre sont illustrés dans un large nombre d'expériences dans le problème de la voiture tout terrain et du jeu de Tetris. Pour le jeu de Tetris, l'utilisation de CBMPI nous a permis de trouver un contrôleur ayant les meilleurs performances rapportées dans la littérature (à notre connaissance). En moyenne, CBMPI obtient des performances supérieures aux méthodes d'entropie croisée (CE), qui sont les solutions de l'état de l'art pour le jeu de Tetris, en utilisant considérablement moins d'échantillons.

Chapitre 5 : Identification des Meilleurs Bras dans un Contexte à Bandits Multiples

- Nous considérons des bandits multiples à bras multiples et étudions le problème d'identifier le meilleur bras dans chaque bandit sous une contrainte de budget fixe.

Nous proposons deux nouveaux algorithmes pour ce problème, appelés exploration des écarts (GapE) et exploration unifiée des écarts (UGapE), qui généralisent l'algorithme UCB-E (Audibert et al., 2010) au contexte multi-bandits. Nous dérivons aussi une variante de ces deux algorithmes, appelée GapE-V et UGapE-V, prenant en compte la variance des bras. Nous étendons l'algorithme UGapE au cas de l'identification des M meilleurs bras pour une précision donnée ϵ.

- Pour chaque algorithme proposé, nous dérivons une borne sur la probabilité d'erreur. Ces bornes sont les premières à avoir été produites dans le contexte multi-bandits. Ces résultats supposent la connaissance d'une quantité caractéristique du problème appelée la *complexité*. Pour dépasser ce problème, nous proposons une version *adaptative* de nos algorithmes dans laquelle la complexité est estimée en-ligne.

- Nous évaluons les performances des algorithmes proposés en utilisant des données synthétiques ainsi que des données obtenues grâce à des essais cliniques. Les résultats empiriques soutiennent nos résultats théoriques en montrant que les algorithmes proposés surpassent les algorithmes de bandits existants n'ayant pas été conçus pour le contexte multi-bandits. Ils montrent aussi que la version adaptative des algorithmes semble capable d'estimer la complexité sans perte globale de performance, et qu'ainsi les algorithmes peuvent être utilisés en pratique en toute sérénité.

Contents

Chapter 1
Introduction

The problem of sequential decision-making under uncertainty arises in everyday life, when we try to find an answer to questions like how to navigate from home to work, how to play and win a game (e.g., backgammon, poker, or the game of Tetris that has been used as an experimental testbed in this thesis), how to retrieve our information of interest from the Internet, how to optimize the performance of a factory, etc. Many interesting sequential decision-making tasks can be formulated as reinforcement learning (RL) problems. In RL, an agent interacts with a dynamic, stochastic, and incompletely known environment with the goal of learning a strategy or *policy* to optimize some measure of its long-term performance (e.g., to remove as many lines as possible in Tetris).

Reinforcement learning has had success in many different domains such as the game of Backgammon (Tesauro, 1994), call admission control (Marbach and Tsitsiklis, 1997), helicopter flight control (Ng et al., 2004), catalog mailing (Simester et al., 2006), and managing spoken dialogue systems (Pietquin et al., 2011). Despite the success of the RL algorithms in a number of different domains, there remain several fundamental obstacles hindering the widespread application of the RL methodology to real-world problems. Such real-world problems are characterized by most, if not all, of the following features:

- Large or even infinite state and/or action spaces. This requires the use of approximation techniques in RL algorithms in order to represent policies and/or value (action-value) functions, i.e., a function that given a policy, maps each state (state-action pair) of the system to a real number that represents the expected performance of the policy when the system starts from that state (state-action pair).

- Training data are expensive. Unlike supervised learning, where large corpora of data are usually available and immediately usable, in RL, learning data are generated by the interaction of the learning agent with the dynamical system (real or simulated) it attempts to control. For complex systems, this results in a considerable overhead for each training sample.

- Requirement for online learning. Offline learning is adequate when the environment is completely stationary, however, this is rarely the case.

- Partial observability. In many problems, the state of the system is not completely or directly measurable by the agent.

In this thesis, we focus on the first two features and try to address their related issues in the context of a relatively novel class of RL algorithms, called classification-based policy iteration (CBPI). CBPI is a variant of approximate policy iteration (API), an approximate dynamic programming (DP) algorithm, that replaces the usual *policy evaluation* (approximating the value or action-value function of the current policy over the entire state or state-action space) and *policy improvement* (generating a new policy with an improved performance from the value or action-value function of the current policy) steps with a learning step in a policy space. To be more precise, the main idea is to replace the policy evaluation step with the rollout estimates of the action-value functions of a finite number of states, called the *rollout set*, and every possible action; and to cast the policy improvement step as a *classification* problem. The training set of this classification problem (in its simplest form) consists of the states in the rollout set as input and the actions with the highest estimate of action-value function at that state as output. This class of API algorithms was first introduced by Lagoudakis and Parr (2003b) and Fern et al. (2004) (and its journal version Fern et al. 2006), and then defined more precisely and fully analyzed by Lazaric et al. (2010a). The theoretical and empirical results of using the CBPI algorithms indicate that they can be used as an alternative to the more standard value function-based API methods (e.g., least-squares policy iteration Lagoudakis and Parr 2003a), and may be potentially advantageous to their value function-based counterparts in problems where good policies are easier to represent, and thus learn, than their value functions (e.g., the game of Tetris, as will be shown in Chapter 4 of this thesis).

In the first part of this thesis, Chapters 3 and 4, we propose two extensions of the existing CBPI algorithms that improve the quality of the rollout estimates of the action-value functions, and as a result, ameliorate the overall performance of the algorithm. For each of these algorithms, we provide a finite-sample performance analysis and empirically evaluate its performance and compare it with similar algorithms. In the second part of the thesis, Chapter 5, we propose methods from the bandits literature to improve the rollout allocation strategy in the CBPI algorithms. The main objective is to allocate a fixed budget of rollouts over the states in the rollout set and the available actions in a way to have the most accurate training set for the classifier. Here we formulate the problem of rollout allocation in CBPI as a class of bandit problems, called *pure exploration*, or more specifically, as a problem in this class, called *multi-bandit best arm identification*, and develop new algorithms with theoretical guarantees for this problem. The application of our proposed algorithms goes beyond the rollout allocation in CBPI, because this class of bandit algorithms is directly related to the important problem of *adaptive resource allocation* that has application in a number of different fields from marketing and advertisement to clinical studies and communication networks.

Contents

1 Motivation

Let us consider the game of Tetris as an example of a sequential decision-making problem under uncertainty. As shown in Figure 1.1, Tetris is a game played on a board originally composed of 20 rows and 10 columns, where pieces of 7 different shapes fall sequentially from the top. Given the current configuration of the board and the new falling piece that together constitute the current state of the game (system), the player (agent) selects an action which corresponds to the choice of the position of the falling piece as well as its rotation. For each filled row (line), a reward of 1 is given, the row is removed and all the cells above it move one line down. The uncertainty in the state transition is due to the randomness in the next falling piece. The goal is to remove as many rows as possible before the game is over, i.e. when the last piece dropped goes above the upper limit of the board. This game constitutes an interesting optimization benchmark in which the goal is to find a controller (policy) that maximizes the average (over multiple games) number of lines removed in a game (score). This optimization problem is known to be computationally hard. It contains a extremely large number of board configurations (about $2^{200} \simeq 1.6 \times 10^{60}$), and even in the case that the sequence of pieces is known in advance, finding the strategy to maximize the score is an NP hard problem (Demaine et al., 2003).

This game can be easily formulated as a Markov decision process (MDP), and thus, dynamic programming (DP) and reinforcement learning (RL) algorithms can be used to solve it (i.e., to find a good or an optimal controller for the game). However, since the problem has a large state space, these algorithms should be used in their approximate form. Most approximate dynamic programming (ADP) and RL algorithms are value function-based, i.e., use a function space to approximate the value (action-value) functions. The quality of the solution generated by these algorithms depends on the choice of the function space (how well it can approximate the value functions of the policies generated in the iterations of the algorithm) and the number of samples used to fit a function (in the selected function space) to each value function. Value function-based ADP and RL algorithms have been applied to the game of Tetris (Tsitsiklis and Van Roy, 1996, Bertsekas and Ioffe, 1996, Farias and Van Roy, 2006, Scherrer, 2013), but unfortunately their performance has been lower by several orders of magnitude than the state of the art techniques that search directly in the

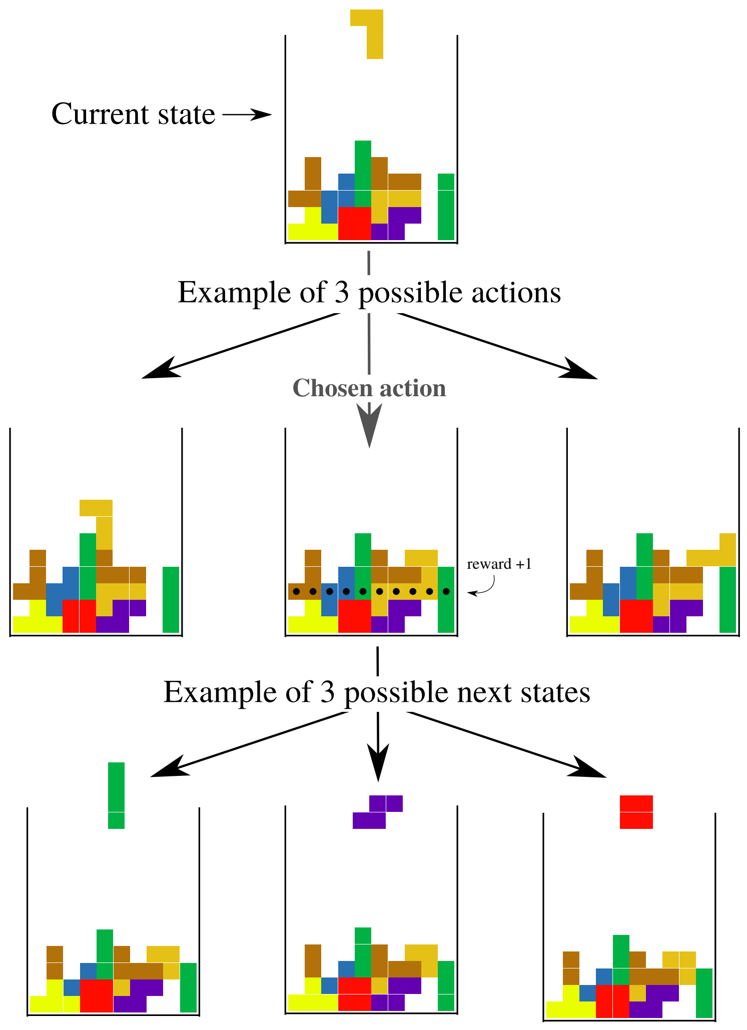

Current state
Example of 3 possible actions
Chosen action
reward +1
Example of 3 possible next states

Figure 1.1: This figure illustrates the state transition in the game of Tetris. The top row shows the current state of the game composed of the current configuration of the board and the falling piece. The second row shows how three different actions affect the configuration of the board. Notably the second action removes a line from the board and collects a reward of 1. Finally, the bottom row shows three possible next states resulted from taking the second action.

space of policies by learning the policy parameters using an optimization black box, such as the cross entropy (CE) method (Szita and Lőrincz, 2006, Thiéry and Scherrer, 2009b). These results support the conjecture that perhaps the game of Tetris is a RL problem in which policies are easier to represent, and thus learn, than their corresponding value functions (e.g., the policies are smooth functions while their corresponding values are very noisy and complex). This makes Tetris a suitable candidate for a relatively novel class of ADP algorithms, called classification-based policy iteration (CBPI) (Lagoudakis and Parr, 2003b, Fern et al., 2004, 2006, Lazaric et al., 2010a).

CBPI algorithms work as follows: at each iteration, given the current policy, **1)** they build a *rollout set* by sampling (using a distribution over states) a number of states from the state space of the problem, **2)** for each state in the rollout set and each action in the action space of the problem, they generate a number of rollouts and compute a rollout estimate of the action-value function at this state-action pair, **3)** they construct a training set with (in its simplest form) the states in the rollout set as input and the action with the highest estimated action-value function as output, and finally **4)** they use this training set to train a classifier, whose output is an estimate of the greedy policy (roughly speaking, a policy whose performance is not worse than the current policy) w.r.t. the current policy. This classifier can be considered as a policy space to approximate the greedy policy. It is important to note that unlike the standard value function-based policy iteration methods that use a function space to approximate the action-value function over the entire state-action space, CBPI estimates the action-value function only at a finite number of state-action pairs. The performance of the CBPI algorithms is directly related to **1)** the quality of the classifier (the richness of the selected policy space), **2)** the accuracy of the generated training set, which in turn depends on the accuracy of the action-value function estimates, and finally **3)** the sampling distribution used to generate the rollout set.

Regarding the accuracy of the action-value function estimates, it is important to note that the rollout estimates of the action-value functions are unbiased (if the rollouts are long enough, i.e. composed of a large number of steps), but may suffer from high variance (the variance increases with the length of the rollout). This raises an important question that: Given a fixed total budget of samples (steps) shared by all the rollouts, how could we generate accurate action-value function estimates in CBPI? This is the question that we try to answer in the first part (Chapters 3 and 4) of this dissertation. In this part, we propose several methods to balance the bias and variance of these estimates and support them with theoretical analysis as well as empirical evidence.

Another important question is: Given the rollout set, how shall we allocate a fixed budget of samples (or rollouts) to these states and the actions in the action space in order to have an accurate training set for the classifier? It is natural to think that it would be more difficult to detect the greedy action (the action with the highest action-value function) at some states than the others, and thus, uniform allocation, which is commonly used, could be wasteful. This question motivates the second part (Chapter 5) of this thesis. We address this question by formulating the problem as a *multi-armed bandit*, and developing bandit algorithms with theoretical guarantees. Therefore, the resulting algorithms are not restricted to the problem of rollout allocation in CBPI and can be used in a more general, and an important, class of problems called *adaptive resource allocation*.

2 Our Approach

As explained above, imposing a budget constraint in CBPI leads to the necessity of truncating the rollouts after a number of steps m. While this reduces the variance of the rollout estimates, it introduces a bias that could affect the performance of the whole algorithm. To address this bias-variance tradeoff, we first propose (in Chapter 3) the use of a value function approximator, called the *critic*, that together with the outcome of the rollout return an estimate of the action-value function. To be more precise, the critic returns an estimate of the value function at the state at which we truncate the rollout. We call the resulting algorithm *direct policy iteration with a critic* (DPI-Critic) after an existing CBPI algorithm called direct policy iteration (DPI) (Lazaric et al., 2010a). The idea is similar to the *actor-critic* algorithms that are among the earliest studied in RL (Barto et al., 1983, Sutton, 1984). If we have a critic with a rich function space, it makes sense to use short rollouts and rely more on the critic, but on the other hand, training a rich critic requires a large number of samples. Similarly for less accurate critics, we need longer rollouts, and thus, more samples for the rollout part. Our theoretical analysis and empirical results of the proposed algorithm indicate that it would be possible to find the right balance between the richness of the critic and the length of the rollouts, given the total budget of samples. This results in the best possible accuracy of the action-value function estimates, given the available budget.

The idea of running a rollout for m steps and then combining its outcome with the value function of the state at which it is truncated is similar to applying the m-step Bellman operator to a given value function. The use of m-step Bellman operator is the trademark of the *modified policy iteration* (MPI) algorithm (Puterman and Shin, 1978) that generalizes the well-known value and policy iteration methods for $m = 1$ and $m = \infty$, respectively. In Chapter 4, we address our question by developing a classification-based version of MPI, called CBMPI, that in addition to a value function approximator (like the standard MPI), uses a classifier to approximate policies. In addition to CBMPI, we developed two approximate MPI (AMPI) algorithms, and for all the algorithms, provided their finite-sample performance analysis (the first for any AMPI algorithm), and evaluated their performance and compared it with similar methods in Chapter 4. Using CBMPI, we found the best reported controller for the game of Tetris. In our experiments, we showed that CBMPI achieves (on average) a performance similar to cross entropy in Tetris (the state of the art in this game) while using significantly less samples.

In the second part of the thesis, Chapter 5, we turn our attention to the second question posed in Section 1, how to allocate rollouts in order to have an accurate training set for the classifier. The accuracy of the training set depends on how successful we are in detecting the greedy action (the action with the highest action-value function) at the states in the rollout set. Note that every time we generate a rollout at a state-action pair, we observe a random sample from a distribution, whose mean is the action-value function at that state-action pair. Therefore, at each state of the rollout set, it is natural to think that we have a number of unknown distributions (equal to the number of possible actions at that state), and the goal is to sample them in a way to detect the one with the highest mean as fast as possible. This problem has been studied in the multi-armed bandit framework under the name *best arm identification* (Maron and Moore, 1993, Bubeck et al., 2009), and several efficient algorithms have been designed for it (Audibert et al., 2010). In this view, each state in the rollout set is a bandit; each available action in that state in an arm; when we pull an arm, we run a rollout and receive a sample from a distribution whose mean is the action-value function of that state-action pair; and the goal is to allocate the available budget (defined in terms of the number of rollouts or pulls) in a way to detect the arm with the largest mean with high probability. However, what is important for us is to detect, as accurately as possible, the greedy action at all the states in the rollout set, and not just at one. Therefore, we need to extend the existing bandit algorithms for best arm identification to multiple bandits. We show in Chapter 5 that this extension is not straightforward or, if straightforward, does not lead to efficient algorithms. We then develop the first algorithms for multi-bandit best arm identification with theoretical guarantees and show their performance in a number of synthetic problems as well as in a problem with clinical data. The problem of best arm identification, both in its single and multi-bandit formulation, is not restricted to the problem of rollout allocation in CBPI. It is directly related to the important problem of *adaptive resource allocation* that has application in a number of different fields from marketing and advertisement to clinical studies and communication networks. It is also referred to as *active learning in multi-armed bandits* and is closely related to the problem of *optimal experimental design* in statistics.

Moreover this version of the problem has other potential applications such as in the following clinical problem. There are M subpopulations, in which one should decide between K_p options for treating subjects from each subpopulation p. A subpopulation may correspond to patients with a particular gene biomarker (or other risk categories) and the treatment options are the available treatments for a disease. The main objective here is to construct a rule, which recommends the best treatment for each of the subpopulations. These rules are usually constructed using data from clinical trials that are generally costly to run. Therefore, it is important to distribute the trial resources wisely so that the devised rule yields a good performance. Since it may take significantly more resources to find the best treatment for one subpopulation than for the others, the common strategy of enrolling patients as they arrive may not yield an overall good performance. Moreover, applying treatment options uniformly at random in a subpopulation could not only waste trial resources, but also it might run the risk of finding a bad treatment for that subpopulation. This problem can be formulated as the *best arm identification* over M multi-armed bandits. In this formulation, each subpopulation is considered as a multi-armed bandit, each treatment as an arm, trying a medication on a patient as a pull, and we are asked to recommend an arm for each bandit after a given number of pulls (budget). The evaluation can be based on **1)** the average over the bandits of the reward of the recommended arms, or **2)** the average probability of error over the bandits (not selecting the best arm), or **3)** the maximum probability of error among the bandits.

3 Contributions

The main contributions of this dissertation are summarized below.

Chapter 3: Introducing a Critic in the CBPI Algorithms

- We propose an algorithm, called *direct policy iteration with a critic* (DPI-Critic), that adds a value function approximation component to the rollout-based action-value function estimation in the CBPI algorithms. To be more accurate, DPI-Critic is an extension of a CBPI algorithm, called *direct policy iteration* (DPI), proposed and analyzed by Lazaric et al. (2010a).

- We theoretically analyze the performance of the DPI-Critic algorithm where the critic is based on the **1)** least-squares temporal-difference learning (LSTD) (Bradtke and Barto, 1996) and **2)** Bellman residual minimization (BRM) (Baird, 1995) methods. For each case, we first show how the error at each iteration of the algorithm propagates through the iterations and then provide a finite-sample bound on its performance after K iterations. The theoretical results indicate that given a fixed budget of samples, depending on several factors, notably the length of the rollouts and the quality of the function approximator, DPI-Critic may achieve a better performance than DPI and least-squares policy iteration (LSPI) (Lagoudakis and Parr, 2003a) algorithms.

- We evaluate the performance of DPI-Critic and compare it with DPI and LSPI in the mountain car and inverted pendulum problems. The empirical results support our theoretical analysis and confirm that DPI-Critic can take advantage of both rollouts and critic and outperform DPI and LSPI.

Chapter 4: Approximate Modified Policy Iteration Algorithms

- We propose three variations of the approximate MPI (AMPI) algorithm, called AMPI-V, AMPI-Q, and CBMPI, that correspond to three well-known approximate DP (ADP) algorithms: fitted value iteration (Munos and Szepesvári, 2008), fitted Q-iteration (Ernst et al., 2005, Antos et al., 2007), and classification-based policy iteration (CBPI), respectively. It is important to note that the classification-based implementation of AMPI (CBMPI) gives another view on this class of algorithms to which the DPI-Critic algorithm, proposed and analyzed in Chapter 3, belongs.

- We report the first error propagation analysis for AMPI that unifies that for approximate policy and value iteration algorithms. We also provide finite-sample performance analysis for the three proposed AMPI algorithms. Our results indicate that the free parameter of MPI allows us to control the balance of errors (in approximating value function and greedy policy) in the final performance of the CBMPI algorithm.

- The performance of CBMPI and the role of its free parameter are illustrated in extensive experiments in the mountain car problem and the game of Tetris. It is important to note that in the game of Tetris, CBMPI finds a controller with the best performance reported in the literature (to the best of our knowledge). On average, it achieves better performance with respect to the cross entropy method (CE), the state of the art solution for Tetris, with significantly less number of samples.

Chapter 5: Multi-bandit Best Arm Identification

- We consider multiple multi-armed bandits and study the problem of identifying the best arm in each bandit under the fixed budget setting. We propose the two new algorithms for this problem, called Gap-based Exploration (GapE) and Unified Gap-based Exploration (UGapE), that are generalization of the UCB-E algorithm (Audibert et al., 2010) to the multi bandit setting. We also derive a variation of these two algorithms, called GapE-V and UGapE-V, that take into account the variance of the arms. We extend the UGapE algorithm to solve the problem of finding the M-best arms with ϵ-accuracy.
- For each proposed algorithm, we derive a bound for its probability of error. These bounds are the first to have been provided for algorithms tackling the multi-bandit setting. These results hold under the assumption that a quantity called the *complexity* of the problem is known. To address this issue, we propose the *adaptive* version of our algorithms in which the complexity is estimated online.
- We evaluate the performance of the proposed algorithms using synthetic data as well as data obtained from clinical trials. The empirical results support our theoretical findings that the proposed algorithms outperform the existing bandit algorithms that have not been designed for the multi-bandit setting. They also show that the adaptive version of the algorithms are able to estimate the complexity without the loss in the global performance, and thus, the algorithms can be safely used in practice.

4 Outline

In this section, we give an outline of the thesis and describe the topics that are covered in each chapter.

- In Chapter 2, entitled "Background and Notations", we provide the background information and introduce the notations necessary to follow the work presented in the thesis. Chapter 2 consists of two parts, one related to reinforcement learning (RL) and one to bandit algorithms (more specifically, to the problem of best arm identification in multi-armed bandits). In the RL part, after introducing the basic concepts and notations, we present exact solutions to the RL problem. We then discuss its approximate solutions, first methods based on approximating the value function and then those that approximate policy. This provides the necessary background to follow our proposed approach, which is a combination of these two approximation schemes. In the bandit part, we first review the classical *cumulative regret* setting before focusing on the more recent, and also more relevant to our work, the *pure exploration* framework. In the latter setting, we explain the best arm identification problem and study it under two different types of constraint, *fixed budget* and *fixed confidence*. Finally, we discuss how our best arm identification algorithms can be used in the CBPI algorithms.

• In Chapter 3, entitled “Classification-based Policy Iteration with a Critic”, we present our first extension to the standard CBPI algorithm. The idea is to estimate the action-value functions as a combination of a truncated rollout and a value function approximator, called *critic*, that approximates the value at the state at which the rollout has been truncated. The role of the critic is to reduce the variance of the action-value function estimates at the cost of introducing a bias. This could improve the overall performance of the CBPI algorithm, especially when we are given a fixed budget of samples. We present a new CBPI algorithm, called *direct policy iteration with a critic* (DPI-Critic), and provide its finite-sample performance analysis when the critic is based on the **1)** least-squares temporal-difference learning (LSTD), and **2)** Bellman residual minimization (BRM) methods. We then empirically evaluate the performance of DPI-Critic and compare it with DPI and LSPI in two benchmark RL problems: mountain car and inverted pendulum.

• In Chapter 4, entitled “Approximate Modified Policy Iteration”, we study the approximate version of the modified policy iteration (MPI) algorithm (Puterman and Shin, 1978). In this chapter, we propose three implementations of approximate MPI (AMPI) that are extensions of well-known approximate DP algorithms. The first two correspond to the classical fitted-value and fitted-Q iteration algorithms, while the last one is based on CBPI. We derive the first error propagation analysis for AMPI that unifies this for approximate policy and value iteration algorithms. We also provide finite-sample performance bounds for all our AMPI algorithms. Finally, we illustrate and evaluate the behavior of our proposed algorithms in the mountain car problem as well as in the game of Tetris.

• In Chapter 5, entitled “Multi-Bandit Best Arm Identification”, we study the problem of identifying the best arm(s) in each of the bandits in a fixed budget multi-bandit multi-armed setting with the objective of designing new efficient algorithms. We first propose two algorithms, called *Gap-based Exploration* (GapE) and *Unified Gap-based Exploration* (UGapE), to solve this problem. Both algorithms focus on the arms with small gap, i.e., arms whose mean is close to the mean of the best arm in the same bandit. We then improve upon these algorithms by introducing GapE-V and UGapE-V, which take into account the variance of the arms in addition to their gaps. We prove an upper-bound on the probability of error for all these algorithms. These are the first algorithms proposed to tackle this problem in the literature that come with theoretical guaranties. Since GapE and GapE-V need to tune an exploration parameter that depends on the complexity of the problem, which is often unknown in advance, we also introduce variations of these algorithms that estimate this complexity online. Finally, we evaluate the performance of these algorithms and compare them to other allocation strategies, first in a number of synthetic problems, second on real-word clinical data, and finally on the CBPI algorithms' rollout allocation problem. It is important to note that the UGapE algorithm allows us to study the link between the two different settings of the best arm identification problem: *fixed budget* and *fixed confidence*.

- In Chapter 6, we summarize the dissertation and discuss some directions for future research.

Chapter 2
Background and Notations

In this chapter, we provide a brief overview of the two main components of this dissertation, namely *reinforcement learning* and *multi-armed bandits*. For each topic, we describe the methods and introduce the notation relevant to our work.

Contents

1 Reinforcement Learning

In this section, we first review the basic setting and definitions of reinforcement learning (RL), then provide an overview of the classical algorithms to solve the RL problem in both exact and approximate forms, and finally motivate and describe in more detail the class of RL algorithms of our interest, namely *classification-based policy iteration* (CBPI), and highlight the related issues that are studied in this thesis.

Before we start with RL, we introduce some mathematical notation that will be used in the RL chapters. For a measurable space with domain $\mathcal{X}$, we let $\mathcal{P}(\mathcal{X})$ and $\mathcal{B}(\mathcal{X}; L)$ denote the set of probability measures over $\mathcal{X}$, and the space of bounded measurable functions with domain $\mathcal{X}$ and bound $0 < L < \infty$, respectively. For a measure $\rho \in \mathcal{P}(\mathcal{X})$ and a measurable function $f : \mathcal{X} \to \mathbb{R}$, we define the $\ell_p(\rho)$-norm of f as $||f||_{p,\rho}^p = \int |f(x)|^p \rho(dx)$.

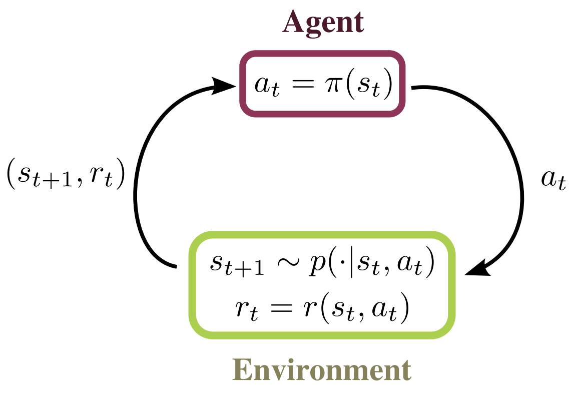


Figure 2.1: The basic loop of agent/environment interaction in the MDP model.

1.1 Markovian Decision Processes and Bellman Operators

As stated in Chapter 1, in RL, an agent interacts with a dynamic, stochastic, and incompletely known environment with the objective of optimizing some measure of its long-term performance. This interaction is often modeled as a Markovian Decision Process (MDP) (Howard, 1960, Puterman, 1994). A discounted MDP $\mathcal{M}$ is a 5-tuple $\langle \mathcal{S}, \mathcal{A}, r, p, \gamma \rangle$ in which

- The **state space** $\mathcal{S}$ is composed of all the possible states (or situations) of the system. This space can be either finite or infinite. For example, in the game of Tetris the state space is typically composed of all the possible configurations of the board and the falling piece. A generic state is denoted by s and by s_t if it is visited at time t.

- The **set of actions** $\mathcal{A}$ contains all the actions available to the agent through which it can interact with its environment. In this thesis, we assume that the action space is **finite**, i.e., $|\mathcal{A}| < \infty$. In Tetris, $\mathcal{A}$ contains all the possible ways that the falling piece can be dropped. A generic action is denoted by a and by a_t if it is taken at time t.

- The **transition model** $p(\cdot|s, a)$ is a distribution over $\mathcal{S}$. It describes the system's dynamics and is the probability distribution over the next state when the agent takes action a in state s. The main property of such a model is that the process is **Markovian**, meaning that the transition probability depends only on the current state and action and is independent of the states and actions previously visited and taken by the agent, i.e., at each time t, $p(\cdot|s_t, a_t) = p(\cdot|s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0, a_0)$.

- The **reward function** $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward obtained by the agent when it takes action a in state s. We assume that the reward is bounded by $R_{\max}$.

- $\gamma \in (0, 1)$ is a **discount factor**.

Therefore, the agent's interaction with the environment can be modeled as a sequence of actions and observations, where at each time-step t, the agent observes the current state of the system s_t, takes an action a_t, receives a reward $r_t = r(s_t, a_t)$, and the system transits, possibly as a results of the action taken by the agent, to state s_{t+1} (see Figure 2.1).

The agent's interaction with the environment at each time-step is often guided by a **policy** that tells the agent which action to take. A deterministic, stationary, Markovian **policy**[1] is defined as a mapping from states to actions, i.e., $\pi : \mathcal{S} \to \mathcal{A}$. Given a policy π, the MDP $\mathcal{M}$ turns to a Markov chain $\mathcal{M}_\pi$ with the reward $r_\pi(s) = r\big(s, \pi(s)\big)$ and transition probability $P_\pi(\cdot|s) = P\big(\cdot\,|s, \pi(s)\big)$.

The main objective in RL is to find (learn) a policy π^* that by following it the agent obtains the maximum expected (discounted) sum of rewards. With this objective in mind, we define the value of policy π in a state s as the expected (discounted) sum of rewards received by starting at state s and following policy π, i.e.,

$$v^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_\pi(s_t) \mid s_0 = s, s_{t+1} \sim P_\pi(\cdot|s_t)\right]. \tag{2.1}$$

The function v^π defined in Equation 2.1 is called the **value function** of policy π. Note that the sum is over an infinite number of terms as we may want to consider infinite horizon problems. Having the discount factor γ strictly smaller than 1, i.e. $\gamma < 1$, guarantees that the value function v^π is bounded by $V_{\max} = Q_{\max} = R_{\max}/(1-\gamma)$. Note that when the problems end in a finite number of steps with probability one (as in the game of Tetris), the discount factor can be set to one.

Two main questions arise immediately from the above definitions: How to compute the value function v^π? and How to use v^π to construct a new policy that is better (not worse) than π? These two questions motivate the following definitions of the Bellman and greedy operators, which play a key role in the standard RL algorithms (see Section 1.2):

The **Bellman operator** T_π of a policy π takes a function $f : \mathcal{S} \to \mathbb{R}$ as input and returns the function $T_\pi f : \mathcal{S} \to \mathbb{R}$ defined as

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

or in compact form, $T_\pi f = r_\pi + \gamma P_\pi f$. It is known that v^π is the unique fixed-point of T_π (see e.g., Bertsekas and Tsitsiklis 1996). Finding v^π can then be done by solving this fixed-point equation, or, as T_π is γ-*contraction* in ℓ_∞-norm, by repetitive application of T_π to an arbitrary initial function.

Given a function $f : \mathcal{S} \to \mathbb{R}$, we say that a policy π is **greedy** with respect to f, and write $\pi = \mathcal{G} f$, if

$$\forall s \in \mathcal{S}, \quad \pi(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \Big\{ r(s,a) + \gamma \mathbb{E}\big[f(s') \mid s' \sim p(.|s,a)\big] \Big\},$$

[1]This is the class of policies that is often studied in the RL literature and also considered in this thesis.

or equivalently $T_\pi f = \max_{\pi'}[T_{\pi'} f]$. The main reason why we are interested in the greedy operator is that it allows us to compute a new policy π which is guaranteed not to be worse than the current policy π', i.e., $v^\pi(s) \geq v^{\pi'}(s)$, $\forall s \in \mathcal{S}$, as $\pi = \mathcal{G}\, v^{\pi'}$. We may combine the above two operators to obtain a new one, called **Bellman optimality operator** T, that takes a value function v as input and directly computes an improved (not worse) value function, i.e., $T : v \to \max_\pi T_\pi v = T_{\mathcal{G}(v)} v$.

We denote by π^* an optimal policy, i.e., a policy with the largest value function, $v^{\pi^*}(s) \geq v^\pi(s)$, $\forall s \in \mathcal{S}$ and $\forall \pi$. On the other hand, we denote by v^* the optimal value function that is the unique fixed-point of the Bellman optimality operator, i.e., $v^* = Tv^*$ (see e.g., Bertsekas and Tsitsiklis 1996). These operator and value function are called optimal because π^* is greedy with respect to v^* and its value satisfies $v^{\pi^*} = v^*$.

The last definition is the **action-value function** of a policy π, $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, that at a state-action pair (s, a) is defined as the expected discounted sum of rewards received by starting at state s, taking action a, and then following policy π,

$$Q^\pi(s,a) = r(s,a) + \gamma \mathbb{E}\Big[v^\pi(s') \mid s' \sim p(.|s,a)\Big] = r(s,a) + \gamma \mathbb{E}\Big[Q^\pi\Big(s', \pi(s')\Big) \mid s' \sim p(.|s,a)\Big].$$

Note that the greedy policy with respect to v^π is simply $\pi(s) = \max_a Q^\pi(s,a)$, $\forall s \in \mathcal{S}$. The main motivation for defining this new quantity comes from the fact that given Q^*, we can easily compute an optimal policy π^* as $\pi^*(s) = \max_a Q^*(s,a)$, $\forall s \in \mathcal{S}$, while given v^*, computing an optimal policy requires the application of the greedy operator to v^*, which in turns requires the knowledge of the system's dynamics.

1.2 Dynamic Programming

In this section, we provide a brief overview of the classical algorithms that solve the RL problem exactly. The approximate case will be the subject of Section 1.3. Here we consider the case where the greedy and Bellman operators can be computed exactly without any approximation or error. This often requires two assumptions: **1)** the state and action space are small enough that we can compute the value (action-value) function exactly (these functions can be represented using relatively small tables) and **2)** the full knowledge of the transition probability and reward of the environment is available so that the operators can be computed without any error. Almost all RL algorithms are instances of one of the two celebrated dynamic programming (DP) algorithms, *policy iteration* and *value iteration*.

• **Policy Iteration:** In Section 1.1, we introduced the Bellman and greedy operators. The former allows us to compute the value function of a policy and the latter to construct a new and improved policy from the value function of the current policy. A natural idea would be to use these operators in alternation in order to have a sequence of monotonically improving policies and eventually find an optimal strategy. This is exactly the basic idea of *policy iteration* (PI) (Howard, 1960). PI starts with an arbitrary initial value function v_0 and generates a sequence of value-policy pairs by computing, at each iteration k, the value function v_{k+1} from v_k in the following manner:

$$\pi_{k+1} = \mathcal{G}\, v_k, \qquad \text{(greedy step)} \qquad (2.2)$$
$$v_{k+1} = (T_{\pi_{k+1}})^{\infty} v_k. \qquad \text{(evaluation step)} \qquad (2.3)$$

In the greedy step, the new policy π_{k+1} is computed by applying the greedy operator to the current value function v_k. In the evaluation step, the value function of the new policy $v_{k+1} = v^{\pi_{k+1}}$ is computed as the fixed point of the Bellman operator of policy π_{k+1}, $T_{\pi_{k+1}}$, by applying this operator infinitely many times (in practice until convergence) to any value function.

- **Value Iteration:** The *value iteration* (VI) (Bellman, 1957) algorithm starts with an arbitrary value function v_0, and at each iteration k, applies the Bellman optimality operator to the current value function v_k, and repeats this process until convergence.

$$v_{k+1} = T v_k. \qquad (2.4)$$

Since the Bellman optimality operator is a combination of the greedy and Bellman operators, VI can be rewritten in the following form that highlights its resemblance to PI.

$$\pi_{k+1} = \mathcal{G}\, v_k \qquad \text{(greedy step)} \qquad (2.5)$$
$$v_{k+1} = (T_{\pi_{k+1}})^{1} v_k \qquad \text{(evaluation step)} \qquad (2.6)$$

- **Modified Policy Iteration:** This algorithm is a natural generalization of PI and VI. While PI applies the Bellman operator of the current policy an infinite number of times (see Equation 2.3) and VI applies it only once (see Equation 2.6), *modified policy iteration* (MPI) (Nunen, 1976, Puterman and Shin, 1978) uses a parameter $m \geq 1$ and applies this operator m times,

$$\pi_{k+1} = \mathcal{G}\, v_k, \qquad \text{(greedy step)} \qquad (2.7)$$
$$v_{k+1} = (T_{\pi_{k+1}})^{m} v_k. \qquad \text{(evaluation step)} \qquad (2.8)$$

MPI has less computation per iteration than PI (in a manner similar to VI), while enjoys the faster convergence (in terms of the number of iterations) of PI, as illustrated in Figure 2.2. MPI is also close to the λ-Policy Iteration (λ-PI) algorithm (Bertsekas and Ioffe, 1996). In λ-PI, the Equation 2.8 of the MPI update rule is replaced by $v_{k+1} = (1-\lambda)\sum_{m=0}^{\infty} \lambda^m (T_{\pi_{k+1}})^{m+1} v_k$. It can be easily shown that λ-PI in fact updates the value function v_k by computing a geometric average of the MPI updates with weight λ^m and parameter $m+1$.

1.3 Approximate Dynamic Programming

In this section, we study a more realistic case, where the Bellman and greedy operators cannot be computed exactly, and it is only possible to approximate them, hence the

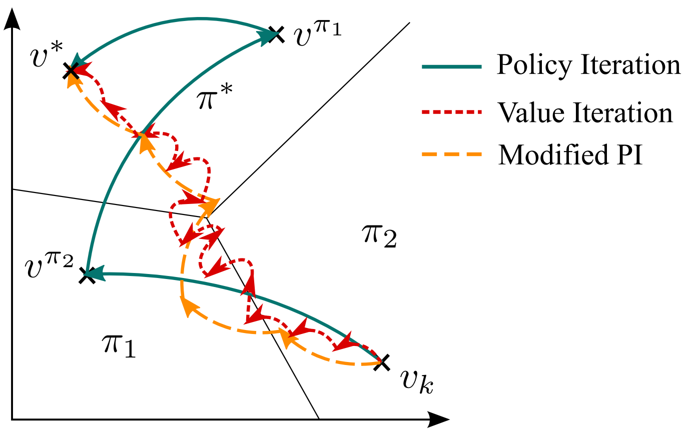


Figure 2.2: A comparison of the VI, PI, and MPI algorithms in terms of the number of iterations needed until convergence. To read this figure, note that the space of value functions has been divided into three areas in which the value functions have the same greedy policy (π_1, π_2, and π^*). At each iteration (each iteration has been represented by an arrow), PI jumps directly from the current value function v to the value function of the greedy policy with respect to v, while MPI, and even more VI, make only a small step in this direction.

name *approximate dynamic programming* (ADP). We further focus only on approximate policy iteration (API), as this is the class of algorithms considered in this thesis. The approximation can be due to the fact that either the system's dynamic is not known completely or the number of states and actions is so large that the value (action-value) functions cannot be stored in a relatively small table. In this thesis, we assume that the system's dynamic is not completely known, but instead we have access to a generative model of the environment from which we can sample. Therefore, we often estimate the value function of a given policy by performing rollouts. A **rollout** is a trajectory that starts at the state of interest s (sometimes followed by the action of interest a) and continues by following a policy π for possibly infinitely many steps. The (discounted) sum of the rewards observed along this trajectory is called the *return* of policy π at state-action pair (s,a),

$$R^\pi(s,a) = r(s,a) + \sum_{t=1}^{\infty} \gamma^t r(s^t, \pi(s^t)), \qquad s^{t+1} \sim p(\cdot|s^t, \pi(s^t)),$$

and is an unbiased estimate of the action-value function $Q^\pi(s,a)$, i.e., $\mathbb{E}[R^\pi(s,a)] = Q^\pi(s,a)$.

To deal with the large number of states and/or actions, we use two possible approximations: **1)** an approximation of the value function in a function space $\mathcal{F}$, and **2)** an approximation of a greedy policy in a policy space Π. In the next two sections, we describe API algorithms based on these two types of approximation, called *value function-based* (or *regression-based*) and *classification-based* policy iteration algorithms, respectively.

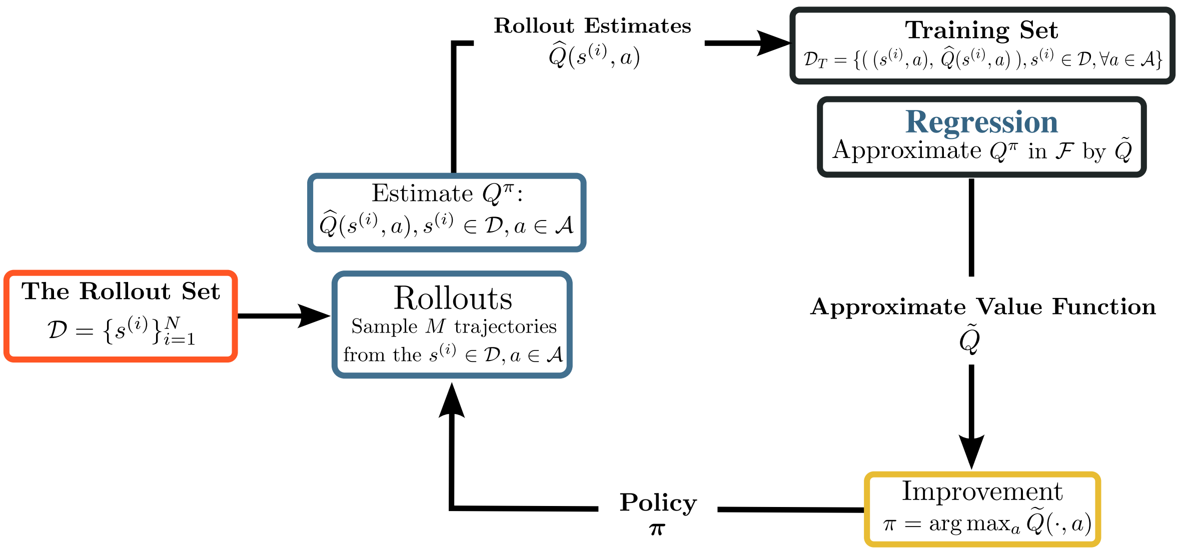

Rollout Estimates
$\widehat{Q}(s^{(i)}, a)$
Training Set
$\mathcal{D}_T = \{(\,(s^{(i)}, a), \widehat{Q}(s^{(i)}, a)\,), s^{(i)} \in \mathcal{D}, \forall a \in \mathcal{A}\}$
Regression
Approximate Q^π in $\mathcal{F}$ by $\tilde{Q}$
Estimate Q^π:
$\widehat{Q}(s^{(i)}, a), s^{(i)} \in \mathcal{D}, a \in \mathcal{A}$
The Rollout Set
$\mathcal{D} = \{s^{(i)}\}_{i=1}^N$
Rollouts
Sample M trajectories from the $s^{(i)} \in \mathcal{D}, a \in \mathcal{A}$
Approximate Value Function
$\tilde{Q}$
Policy
π
Improvement
$\pi = \arg\max_a \widetilde{Q}(\cdot, a)$

Figure 2.3: The flowchart of the value function-based PI algorithm of Section 1.3.1.

1.3.1 Value Function-based (Regression-based) Policy Iteration

In this section, we give the basic idea of this class of API methods and briefly describe several such algorithms.

• **Linear Approximation:** The first step in these algorithms is to define a function space $\mathcal{F}$ to approximate the value functions. A natural and widely-used candidate for $\mathcal{F}$ is the linear space spanned by a relatively (with respect to the size of the state space) small number of basis functions. We consider a linear architecture with parameters $\alpha \in \mathbb{R}^d$ and bounded (by L) basis functions $\{\varphi_j\}_{j=1}^d$, $\|\varphi_j\|_\infty \leq L$. We denote by $\phi : \mathcal{X} \to \mathbb{R}^d$, $\phi(\cdot) = \big(\varphi_1(\cdot), \dots, \varphi_d(\cdot)\big)^\top$ the feature vector, and by $\mathcal{F}$ the linear function space spanned by the features φ_j, i.e.,

$$\mathcal{F} = \{f_\alpha(\cdot) = \phi(\cdot)^\top \alpha : \alpha \in \mathbb{R}^d\}.$$

We define the Gram matrix $G \in \mathbb{R}^{d\times d}$ with respect to a distribution μ over $\mathcal{S}$ as

$$G_{ij} = \int_{\mathcal{S}} \varphi_i(s)\varphi_j(s)\mu(ds), \qquad i, j = 1, \dots, d. \tag{2.9}$$

• **Algorithms:** Approximate PI (API) has been the focus of a rich literature (see e.g., Bertsekas and Tsitsiklis 1996, Szepesvári 2010).[2] API first finds an approximation of the value function of the current policy and then generates the next policy as the greedy policy with respect to this approximation (Bertsekas and Tsitsiklis, 1996, Munos, 2003, Lagoudakis and Parr, 2003a). A related algorithm is λ-policy iteration (Bertsekas and Ioffe, 1996), which is a rather complicated variation of MPI, as

[2]Similarly approximate value iteration (AVI) that at each iteration generates the next value function as the approximation of the application of the Bellman optimality operator to the current value function (Ernst et al., 2005, Antos et al., 2007, Munos and Szepesvári, 2008)

discussed in Section 1.2. This algorithm involves computing a fixed-point at each iteration and has been analyzed in its approximate form by Thiéry and Scherrer (2010a) (see also Scherrer 2013).

Let us now discuss a simple value function-based API algorithm in detail. Figure 2.3 shows the structure of this algorithm. This is an iterative algorithm in which at each iteration, **1)** a rollout set is generated, $\mathcal{D} = \{s^{(i)}\}_{i=1}^N$, composed of N states sampled i.i.d. from some distribution over the states, **2)** for each state $s^{(i)} \in \mathcal{D}$ and each action $a \in \mathcal{A}$, an estimate, $\widehat{Q}(s^{(i)}, a)$, of the action-value function of the current policy, $Q^\pi(s^{(i)}, a)$, is computed by performing M rollouts starting from state-action pair $(s^{(i)}, a)$ and following policy π, **3)** from these estimates, $\widetilde{Q}$ is computed as an approximation of Q^π, in the function space $\mathcal{F}$, using e.g., the least-squares method, and finally **4)** the new policy is computed as the greedy policy with respect to the approximate action-value function $\widetilde{Q}$, simply as $\pi(s) = \operatorname{argmax}_a \widetilde{Q}(s,a)$, $\forall s \in \mathcal{S}$.

This class of API algorithms can fail if the number of samples is not large enough and/or the approximation space $\mathcal{F}$ is not rich enough. However, there are problems in which good or optimal value functions are more difficult to represent, and thus, learn than their corresponding policies. This is the main motivation for introducing a new class of API algorithms, called classification-based PI, in the next section.

1.3.2 Classification-based Policy Iteration

In Section 1.3.1, the policies were implicitly defined by value functions. That is, the performance of the algorithms depends on the quality of value function approximation. An alternative approach would be not to use an explicit approximation for the value function and instead to learn an approximation of the greedy policy using a policy space. This is the approach taken by the family of classification-based PI (CBPI) algorithms, which are the main API methods studied in this dissertation. As discussed earlier, this approach is expected to work better than the regression-based PI methods in problems in which good or optimal polices are easier to approximate and learn than their corresponding value functions. A simple example is the mountain car domain (see Section 5.1 in Chapter 3 for a detailed description) where the objective is to drive a car up a hill by accelerating either towards the right or the left. In this domain, the optimal value function, which takes the position and the velocity of the car as inputs, has a complex shape (see for instance Example 8.2 in Sutton and Barto (1998)) while there exists an excellent policy which is simply a linear separator in the position/velocity space between a region where one should accelerate right (when the velocity is positive) or accelerating left (otherwise). The CBPI algorithms do not compute the greedy policy using an explicit approximation of the value (action-value) function but estimate it as the output of a classifier (the policy space Π is defined by the classifier). CBPI algorithms were first proposed by Lagoudakis and Parr (2003b) and Fern et al. (2004) (see also the longer version of this work, Fern et al. 2006). Lazaric et al. (2010a) introduced an algorithm, called direct policy iteration (DPI), and provided its finite-sample analysis, the first full analysis of this kind for this class of algorithms. In their analysis, they theoretically showed the importance of using cost sensitive classification

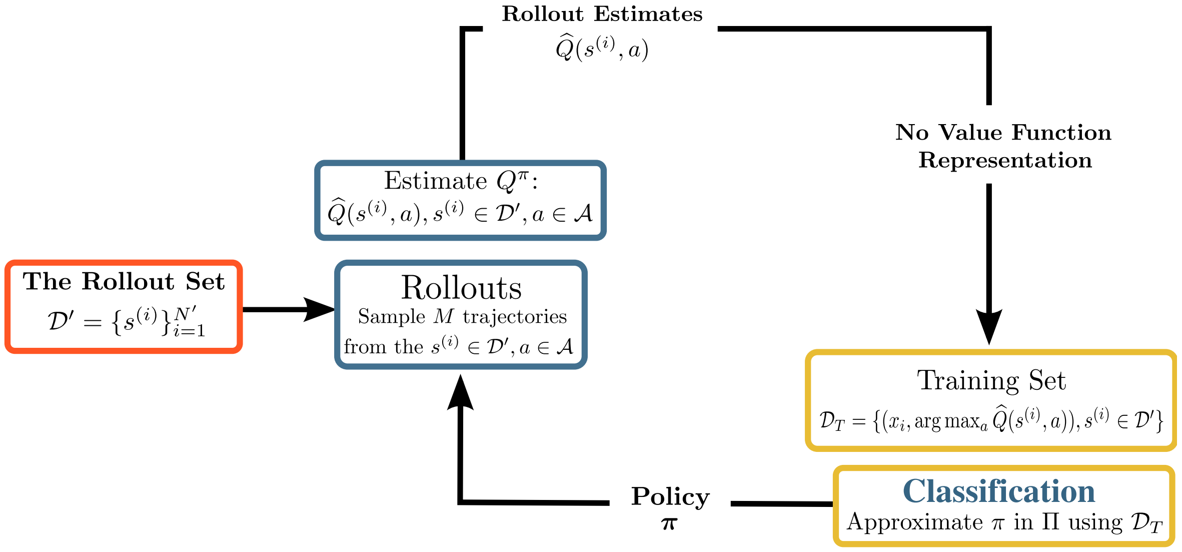

Rollout Estimates
$\widehat{Q}(s^{(i)}, a)$
No Value Function Representation
Estimate Q^π: $\widehat{Q}(s^{(i)}, a), s^{(i)} \in \mathcal{D}', a \in \mathcal{A}$
The Rollout Set $\mathcal{D}' = \{s^{(i)}\}_{i=1}^{N'}$
Rollouts
Sample M trajectories from the $s^{(i)} \in \mathcal{D}', a \in \mathcal{A}$
Training Set
$\mathcal{D}_T = \{(x_i, \arg\max_a \widehat{Q}(s^{(i)}, a)), s^{(i)} \in \mathcal{D}'\}$
Policy π
Classification
Approximate π in Π using $\mathcal{D}_T$

Figure 2.4: The flowchart of the classification-based PI algorithm of Section 1.3.2.

to estimate the greedy policy.

Figure 2.4 shows the flowchart of a CBPI algorithm. In this algorithm, **1)** a rollout set is built, $\mathcal{D}' = \{s^{(i)}\}_{i=1}^{N'}$, composed of N' states sampled i.i.d. from some distribution over the states, **2)** for each state $s^{(i)} \in \mathcal{D}'$ and each action $a \in \mathcal{A}$, an estimate, $\widehat{Q}(s^{(i)}, a)$, of the action-value function of the current policy $Q^\pi(s^{(i)}, a)$, is computed by performing M rollouts starting from state-action pair $(s^{(i)}, a)$ and following policy π, (so far everything is similar to the algorithm described in Section 1.3.1), now unlike the algorithm in Section 1.3.1, the rollout estimates are not used in order to approximate the value (action-value) function over the entire state (state-action) space, but instead **3)** they are used to build a training set that is fed to a classifier, whose output is an approximation of the greedy policy with respect to π. This classifier minimizes the following cost-sensitive error function:

$$\widehat{\mathcal{L}}^\Pi(\pi) = \frac{1}{N'} \sum_{i=1}^{N'} \Big[\max_{a \in \mathcal{A}} \widehat{Q}(s^{(i)}, a) - \widehat{Q}\big(s^{(i)}, \pi(s^{(i)})\big) \Big]. \tag{2.10}$$

This error function is the result of the cost-sensitive loss function introduced and analyzed by Lazaric et al. (2010a). It would be possible to use the following error function resulted from a 0/1 loss function:

$$\widehat{\mathcal{L}}^\Pi(\pi) = \frac{1}{N'} \sum_{i=1}^{N'} \Big[\mathbb{I}\{\pi(s^{(i)}) \neq \operatorname*{argmax}_{a \in \mathcal{A}} \widehat{Q}(s^{(i)}, a)\} \Big]. \tag{2.11}$$

The idea of the cost-sensitive loss function is to penalize a suboptimal action relative to the difference between its action-value function and the action-value function of the greedy action. This is different from the 0/1 loss function that penalizes all suboptimal actions equally.

• **Conservative PI:** Kakade and Langford (2002) proposed an API algorithm that at each iteration computes the new policy as a combination of the approximate greedy policy computed by the classifier and the policies calculated at the previous iterations. A nice property of this algorithm is its monotonically improving behavior which is different than almost all other ADP methods that converge to a region in general. However, Ghavamzadeh and Lazaric (2012) recently theoretically compared the conservative PI algorithm with DPI and showed that the desired property of the conservative PI algorithm does not come for free. They showed that in order to achieve the same level of accuracy, the conservative PI algorithm requires more iterations, and thus, more samples than DPI. This indicates that although this algorithm's conservative update allows it to have a monotonically improving behavior, it slows down the algorithm and increases its sample complexity. Furthermore, this algorithm may converge to suboptimal policies whose performance is not better than those returned by DPI. Finally, it requires memory to store all the policies generated at the iterations of the algorithm.

• **Rollout Allocation:** As described above, in standard CBPI, we allocate the rollouts uniformly over the states in the rollout set $\mathcal{D}'$ and the actions in $\mathcal{A}$. These rollouts are used to estimate the action-value functions that in turn define the loss function of the classifier (see Equations 2.10 and 2.11), and ultimately allow us to approximate the greedy policy. However, one may argue that the uniform allocation of the rollouts is not the optimal way to achieve this final objective. For example, in the case of 0/1 loss (Equation 2.11), in order to provide an accurate training set for the classifier, the objective of the rollouts should be to identify the action with the highest action-value function at each rollout state. Therefore, instead of allocating the rollouts uniformly over actions, they should be allocated more to the actions that are likely to be the best at a particular state. Moreover, the states at which it is more difficult to detect the best action require more rollout, and thus, uniform allocation over states may not be optimal. In the case of the cost-sensitive loss (Equation 2.10), it is not clear at first what a "smart" allocation should be, however, allocating for instance more rollouts to the state-action pairs whose associated return has large variance would clearly lead to an improvement over uniform allocation.

The above remarks indicate that we need smarter (than uniform) strategies for rollout allocation in CBPI. As discussed above, a good allocation should depend on the unknown values of the state-actions pairs, and thus, it should be adaptive. As we will discuss in Section 2, the rollout allocation in CBPI can be viewed and cast as an instance of the multi-armed bandit problem.

• **Rollout Truncation and the Bias-Variance Trade-off:** In the standard CBPI formulation, it is assumed that the rollouts are of infinite length (if we do not reach a terminal state). This assumption causes two problems in practice: **1)** the estimates computed using infinite (long) rollouts are subject to a large variance, as they are composed of a (discounted) sum of infinitely many random variables, and **2)** performing infinite (long) rollouts is simply impossible, because we only have access to limited computation and number of samples. For these reasons, rollouts are usually truncated after m steps. This is equivalent to assuming that the sum of the rewards is zero after the m-th step. This introduces a bias in the rollout estimates that grows as we decrease m. As a result, having long rollouts means less bias and more variance and having short rollouts is equivalent to having more bias and less variance. This calls for a bias-variance trade-off and developing methods to find good values for m.

• **Rollout Trade-off – Tuning Parameters M, N', m, Given a Fixed Budget:** Without any time or sample constraints, we should choose to have large M (more rollouts to reduce the variance of the estimates), large m (longer rollouts to reduce the bias of the estimates), and large N' (more states to feed to the classifier) in order to obtain a good performance. Since this is not realistic, we should study how to tune these parameters under a budget constraint, i.e., a constraint on the number of possible calls to the generative model B. Although Lazaric et al. (2010a) showed that **1)** when m is fixed, it is theoretically better to set M to 1 and to favor large number of states in the rollout set (large N'), and **2)** when M is fixed, m should be of order $\frac{\log B}{\log 1/\gamma}$, tuning these three parameters is non-trivial in general.

In this thesis, we address this **rollout trade-off** in order to improve the quality of the training set of the classifier. One approach that we propose is to use a critic (a value function approximator) in CBPI that provides estimates of the sum of the rewards that could have been collected after the truncation of the rollouts. The hope is that using a critic will reduce the bias caused by the truncation of the rollouts and even permit to control the variance of the rollout estimates as it allows us to rely on shorter rollouts. As discussed in Chapter 1, the idea of this approach, using separate representations for policy and value function, is similar to the idea of the **actor-critic** algorithms (Barto et al., 1983, Sutton, 1984).

2 Multi-Armed Bandit Problems

In this section, we first review the basic definitions of multi-armed bandits, provide a brief overview of the classical *cumulative regret* setting and motivate the *pure exploration* framework that allows us to tackle the resource allocation problem arises in the CBPI algorithms.

2.1 Preliminaries

The multi-armed bandit problem is a general framework where a forecaster plays a repetitive game in which, at every time step t, he chooses a distribution, among K distributions, to sample from. There are different ways to measure the performance of the forecaster in this problem that will be discussed in this section. This framework is used in many resource allocation problems, and thus, fits the problem of rollout allocation in the classification-based policy iteration (CBPI) algorithms (see the previous section for more detail).

Before discussing different performance measures in bandits and link it to CBPI, we introduce the bandit notation used throughout this dissertation. Let $A = \{1, \dots, K\}$ be the set of arms such that each arm $k \in A$ is characterized by a distribution ν_k bounded in $[0, b]$ with mean μ_k and variance σ_k^2. We define the m-max and m-argmax operators as[3]

$$\mu_{(m)} = \overset{m}{\max_{k \in A}} \; \mu_k \qquad \text{and} \qquad (m) = \arg \overset{m}{\max_{k \in A}} \; \mu_k \; ,$$

where (m) denotes the index of the m-th best arm in A and $\mu_{(m)}$ is its corresponding mean so that $\mu_{(1)} \geq \mu_{(2)} \geq \ldots \geq \mu_{(K)}$. We denote by $S^m \subset A$ any subset of m arms (i.e., $|S^m| = m < K$) and by $S^{m,*}$ the subset of the m best arms (i.e., $k \in S^{m,*}$ if $\mu_k \geq \mu_{(m)}$). Without loss of generality, we assume that there exists a unique set $S^{m,*}$. In the following we drop the superscript m and use $S^* = S^{m,*}$ whenever m is clear from the context. With a slight abuse of notation we further extend the m-max operator to an operator returning a set of arms, such that

$$\{\mu_{(1)}, \dots, \mu_{(m)}\} = \overset{1..m}{\max_{k \in A}} \; \mu_k \qquad \text{and} \qquad S^* = \arg \overset{1..m}{\max_{k \in A}} \; \mu_k \; .$$

For each arm $k \in A$, we define the gap Δ_k as

$$\Delta_k = \begin{cases} \mu_k - \mu_{(m+1)} & \text{if } k \in S^*, \\ \mu_{(m)} - \mu_k & \text{if } k \notin S^*. \end{cases}$$

This definition of gap indicates that if $k \in S^*$, Δ_k represents the "advantage" of arm k over the suboptimal arms, and if $k \notin S^*$, Δ_k denotes how suboptimal arm k is. Note that we can also write the gap as $\Delta_k = |\overset{m}{\max_{i \neq k}} \mu_i - \mu_k|$. Given an accuracy ϵ and a number of arms m, we say that an arm k is (ϵ,m)-optimal if $\mu_k \geq \mu_{(m)} - \epsilon$.

The multi-armed bandit problem can be formalized as a game between a stochastic environment and a forecaster. The distributions $\{\nu_k\}$ are unknown to the forecaster. At each round t, the forecaster pulls an arm $I(t) \in A$ and observes an independent sample drawn from the distribution $\nu_{I(t)}$. Depending on the measure of performance, the forecaster may want to have an empirical estimate (using the observed samples) of the expected value or variance of each arm. Let $T_k(t)$ be the number of times that arm k has been pulled by the end of round t, then the empirical mean and variance of this arm are calculated as $\widehat{\mu}_k(t) = \frac{1}{T_k(t)} \sum_{s=1}^{T_k(t)} X_k(s)$ and $\widehat{\sigma}_k^2(t) = \frac{1}{T_k(t)-1} \sum_{s=1}^{T_k(t)} \Big(X_k(s) - \widehat{\mu}_k(t)\Big)^2$, where $X_k(s)$ is the s-th sample observed from ν_k.

In the following two sections, we discuss two formulations of the stochastic bandit problem. The first formulation corresponds to the classical cumulative regret setting where the forecaster tries to constantly choose the distribution with the highest mean (Section 2.2). The second one is the "pure exploration" setting, where the forecaster uses the exploration phase to find a solution according to which he will be evaluated at the end of this phase (Section 2.3).

[3]Ties are broken in an arbitrary but consistent manner.

2.2 Cumulative Regret Setting

The cumulative regret setting is the standard formulation for multi-armed bandits. It is motivated by many applications such as the adaptive routing problem. In this problem, the forecaster needs to choose between K different possible paths to send messages from A to B in a network. The goal is to use the fastest path as frequently as possible. However, the time taken by a communication in a path is not fixed as it depends on several random events such as the global traffic of the network at time t. Moreover, the forecaster does not know the expected time of each path and needs to test the paths sufficiently enough in order to have an accurate estimate of this quantity. The more a path is used the better its time estimate would be. Therefore, at a time t, the forecaster faces a trade-off between using the path that seems to be the fastest according to the current estimates (*exploitation*) and testing other paths to see whether they are better than that one (*exploration*). This is called the exploration/exploitation trade-off. This trade-off is extremely important in sequential decision making under uncertainty, since often in these problems, the forecaster is repetitively given the choice between the available options, but information about an option can be gathered only if it is selected. This problem can have many variants depending on the structure of the decisions and/or the nature of the feedback given to the forecaster.

In multi-armed bandit theory, the most standard variant is the *stochastic* case that corresponds to the case where all the samples generated i.i.d. (independent and identically distributed) from the K distributions.[4] In this formulation, the objective is to minimize the expected cumulative regret defined as

$$R(n) = n\mu_{(1)} - \mathbb{E}\left[\sum_{t=1}^{n} \mu_{I(t)}\right].$$

This problem was introduced by Robbins (1952). Auer et al. (2002) proposed the UCB algorithm, a very simple and efficient solution to this problem based on the use of *Upper Confidence Bounds*. At each time t, UCB selects the arm with the highest high-probability upper-bound on its mean. For an arm k, this upper confidence bound is defined by the index

$$\hat{\mu}_k(t) + \sqrt{\frac{2\log(t)}{T_k(t-1)}},$$

which is composed of the empirical average of the arm, $\hat{\mu}_k(t)$, plus an exploration term that is the radius of the confidence interval on the first moment of the distribution (its mean). This exploration term is derived from the Chernoff-Hoeffding concentration inequality. While the Chernoff-Hoeffding inequality does not take into account the higher moments of the distribution, other algorithms based on more advanced concentration inequalities have been proposed. Audibert et al. (2007) extended the UCB algorithm with the UCB-V (UCB-Variance) algorithm that takes into account the variance of the arms. Cappé et al. (2013) extended this approach by using an even tighter concentration inequality based on the Kullback-Leibler divergence. On the

[4]Throughout this thesis, we also make the assumption that the samples are i.i.d..

other hand, Kaufmann et al. (2012) provided a distribution dependent analysis of the Thomson sampling algorithm, a Bayesian method which has been shown to be efficient in practice (Chapelle and Li, 2011).

The bandit problem has also been studied where the generated samples from each arm are not i.i.d., but instead chosen by an adversary (Auer et al., 2003). In this case, the goal of the forecaster is to limit its expected regret with respect to the best constant strategy. Another possible way to extent the initial formulation is to consider cases with an infinite number of arms. The case where no structure is assumed on the set of arms has been considered by Wang et al. (2008) under a stochastic assumption. When the bandits are in a linear structure, Abbasi-Yadkori et al. (2011) tackled the case of stochastic rewards while Dani et al. (2007) tackled the adversarial case.

2.3 Pure Exploration Setting

The pure exploration is a relatively new setting, where the forecaster is only evaluated at the end of an exploration phase. Contrary to the cumulative regret setting, the rewards collected before the end of the game are not taken into account. This breaks the traditional exploration-exploitation trade-off as it seemingly removes the need to exploit.

This type of scenario happens for instance when a budget is allocated to answer a particular question about the distributions of the arms. For instance, Antos et al. (2010) studied the case where a company wants to conduct a sequence of inspections in order to assess the quality of all its K machines with equal precision. A machine is seen as an arm and an inspection can be seen as a random sample generated from an underlying distribution of this arm giving a noisy measurement (estimate) of the quality of the machine. The more samples (inspections) are collected from a machine the more precisely its quality is assessed. However, depending on the magnitude of the noise in the measurements, the quality of some machines can be assessed very precisely using only a few inspections (in the case of a small noise), while for others it may require many more measurements (when the noise is large). For example, if the measure of performance is the maximum expected squared error after n pulls, $\max_{i\in\{1,\dots,K\}} \mathbb{E}\left[(\mu_i - \widehat{\mu}_i(n))^2\right]$, then the optimal allocation that minimizes this error pulls each arm proportionally to its variance. For this problem, Carpentier et al. (2011) proposed a new algorithm that pulls at each time step the arm with the highest upper confidence bound on its variance.

From this common use of upper-bounds, it can be noticed that the pure exploration strategies look very similar to the ones of the cumulative regret setting that balance exploration and exploitation. The main difference is that the radius of the confidence intervals used in the indices of pure exploration algorithms are usually tuned with respect to the parameters specified by the problem. We now introduce the particular pure exploration problem that is of our interest for the study of the resource allocation in CBPI.

2.3.1 Simple Regret & Best Arm Identification

The problem of *best arm(s) identification* (Even-Dar et al., 2006, Bubeck et al., 2009, Audibert et al., 2010) in the stochastic multi-armed bandit setting has recently received much attention. In this problem, the forecaster is asked to return the best arm(s) among K at the end of the exploration phase and is evaluated on the quality of the recommended arm(s). This abstract problem models a wide range of applications. For instance, let us consider a company that has K different variants of a product and needs to identify the best one(s) before actually placing it on the market. The company sets up a testing phase in which the products are tested by potential customers. Each customer tests one product at a time and gives it a score (a reward). The objective of the company is to return a product at the end of the test phase which is likely to be successful once placed on the market (i.e., the best arm identification), and it is not interested in the scores collected during the test phase (i.e., the cumulative reward). More formally, the objective in the best arm identification problem is to return m (ϵ,m)-optimal arms. This objective has been studied under two different settings, namely *fixed budget* and *fixed confidence*. Before defining these settings, we define the notion of *simple regret* for each arm $k \in A$ as

$$r_k = \mu_{(m)} - \mu_k, \tag{2.12}$$

and for any set $S \subset A$ of m arms, we define the *simple regret* as

$$r_S = \max_{k\in S} r_k = \mu_{(m)} - \min_{k\in S} \mu_k. \tag{2.13}$$

We denote by $\Omega(t) \subset A$ the set of m arms returned by the forecaster at the end of the exploration phase (when the algorithm stops after t rounds), and by $r_{\Omega(t)}$ its corresponding simple regret. Returning m (ϵ,m)-optimal arms is then equivalent to having $r_{\Omega(t)}$ smaller than ϵ. Other performance measures can be defined for this problem. In some applications, returning the wrong arm is considered as an error independently from its regret, and thus, the objective is to minimize the average probability of error

$$\ell(t) = \mathbb{P}\Big(\Omega(t) \neq S^*\Big). \tag{2.14}$$

This corresponds to the problem of returning m (ϵ,m)-optimal arms for the particular case where $\epsilon = 0$. It is interesting to note the relationship between these two performance measures: $\min_k \Delta_k \times \ell(n) \leq \mathbb{E}[r(n)] \leq b \times \ell(n)$, where the expectation in the regret is with respect to the random samples. As a result, any algorithm minimizing the probability of error, $\ell(n)$, also controls the simple regret $\mathbb{E}[r(n)]$. Most of the algorithms in the literature directly target the problem of minimizing $\ell(n)$.

• **Complexity:** The hardness of the best arm identification problem in the fixed budget and the fixed confidence settings is captured by the quantities H and $\overline{H}$ defined as

$$H = \sum_k \frac{b^2}{\Delta_k^2}, \qquad \text{and} \qquad \overline{H} = \max_k \frac{k}{\Delta_{(k)}^2}, \tag{2.15}$$

in the case where $\epsilon = 0$. H and $\overline{H}$ are connected by the following relation: $\overline{H} \leq H \leq \overline{H}\log(K)$. Intuitively H and $\overline{H}$ can be interpreted as the total number of pulls required to discriminate the best arm(s) from the others and will naturally appear in the theoretical results of most of the considered algorithms.

Given an accuracy ϵ and a number of arms m to return, we now formalize the two settings of the (ϵ,m)-best arm identification problem, *fixed budget* and *fixed confidence*.

- **Fixed Confidence:** In the *fixed confidence* setting (see e.g., Maron and Moore 1993, Even-Dar et al. 2006), the forecaster tries to minimize the number of rounds needed to achieve a fixed confidence on the quality of the returned best arm(s). In the above example, the company keeps enrolling customers in the test until it is, e.g., 95% confident that the best product has been identified. More precisely, the goal is to design a forecaster that stops as soon as possible and returns a set of m (ϵ,m)-optimal arms with a fixed confidence. We denote by $\widetilde{n}$ the time when the algorithm stops and by $\Omega(\widetilde{n})$ its set of returned arms. Given a confidence level δ, the forecaster has to guarantee its policy is (m, ϵ, δ)-correct, meaning that $\mathbb{P}\Big[r_{\Omega(\widetilde{n})} \geq \epsilon\Big] \leq \delta$. The performance of the forecaster is then measured by the number of rounds $\widetilde{n}$ either in expectation or high probability.

Maron and Moore (1993) considered a slightly different setting where besides a fixed confidence the maximum number of rounds is also fixed. They designed an elimination algorithm for the case $m = 1$, called Hoeffding Races, based on progressively discarding the arms that are suboptimal with enough confidence. Mnih et al. (2008) introduced an improved algorithm, built on the Bernstein concentration inequality, which takes into account the empirical variance of each arm. Even-Dar et al. (2006) studied the *fixed confidence* setting without any budget constraint. They designed several elimination algorithms and provided the analysis to bound with high probability the stopping time of these algorithms. Mannor and Tsitsiklis (2004) proved a lower bound showing that any $(m = 1, \epsilon = 0, \delta)$-correct policy has an expected stopping time at least of order of $H\log(\frac{1}{\delta})$. Finally, Karnin et al. (2013) proposed an algorithm, called Exponential-Gap, for which they proved an upper bound on its required number of pulls within a gap of $\log\log(\frac{1}{\Delta_{min}})$ of the lower bound of Mannor and Tsitsiklis (2004).

Kalyanakrishnan and Stone (2010) provided different heuristics for the case where the m-best arms must be returned with a given confidence, with m possibly bigger than 1. A thorough theoretical analysis was then provided in Kalyanakrishnan et al. (2012) for the newly introduced algorithm called LUCB (based on the use of *Lower and Upper Confidence Bounds*). Recently, Kaufmann and Kalyanakrishnan (2013) proposed KL-LUCB, an extension of LUCB, which takes into account the empirical Kullback-Leibler divergence between the arms in the case of Bernoulli distributions.

- **Fixed Budget:** In the *fixed budget* setting (see e.g., Bubeck et al. 2009, Audibert et al. 2010), the number of rounds of the exploration phase is fixed and is known by the forecaster, and the objective is to maximize the probability of returning the best

```
**Parameters:** number of rounds $n$, maximum range $b$, complexity $H$
**Initialize:** $T_k(0) = 0$, $\widehat{\mu}_k(0) = 0$ for all arm $k \in A$
**for** $t = 1, 2, \ldots, n$ **do**
    Draw $I(t) \in \arg\max_k \widehat{\mu}_k(t-1) + b\sqrt{\frac{n}{HT_k(t-1)}}$
    Observe $X_{I(t)}(T_{I(t)}(t-1)+1) \sim \nu_{I(t)}$
    Update $T_{I(t)}(t) = T_{I(t)}(t-1)+1$ and $\widehat{\mu}_k(t)$ $\forall k$ of the selected bandit
**end for**
Return $\Omega(n) \in \text{argmax}_{k \in A} \widehat{\mu}_k(n)$
```

Figure 2.5: The pseudo-code of the UCB Exploration (UCB-E) algorithm.

arm(s). In the above example, the company fixes the length of the test phase before hand (e.g., enrolls a fixed number of customers) and defines a strategy to choose which products to show to the testers so that the final selected product is the best with the highest probability. More precisely, the objective is to design a forecaster capable of returning a set of m (ϵ,m)-optimal arms with the largest possible confidence using a fixed budget of n rounds. More formally, given a budget n, the performance of the forecaster is measured by the probability $\widetilde{\delta}$ of not meeting the (ϵ,m) requirement, i.e., $\widetilde{\delta} = \mathbb{P}\Big[r_{\Omega(n)} \geq \epsilon\Big]$, the smaller $\widetilde{\delta}$, the better the algorithm.

Audibert et al. (2010) proposed two different strategies to solve this problem in the case where $m = 1$ and $\epsilon = 0$. They defined a strategy based on upper confidence bounds, called UCB-E (see Figure 2.5 for its pseudo-code), which is very similar to the UCB algorithm explained in Section 2.2. The only difference of UCB-E and UCB is that the optimal parameterization of its exploration term is related to the length of the exploration phase n and to the complexity of the problem H. Note that H is not usually known in advance. The authors proved that the probability of error of UCB-E was bounded by a term of order $nK\exp(-\frac{n}{H})$. They also introduced an elimination algorithm, called Successive Rejects (SR), which divides the budget n in phases and discards one arm per phase. Contrary to UCB-E, SR is parameter free and do not require the knowledge of the complexity H. This comes to the cost of a slightly worse bound of order $K^2\exp(-\frac{n}{\overline{H}\log(K)})$. On the other hand they showed a lower bound of order $\exp(-\frac{n}{H})$ for the problem. Finally, their experimental results show that in the fixed budget setting the algorithms designed specifically for this setting (and especially UCB-E) largely outperform the Hoeffding Races, which were designed for the fixed confidence setting.

Karnin et al. (2013) have designed a new version of the SR algorithm that removes half of the remaining arms at the end of each phase (while the original SR removes one at a time) with improved and new upper bounds on the probabilities of error of order $\log(K)\exp(-\frac{n}{\overline{H}\log(K)})$. Recently, Wang et al. (2013) extended the SR results to the problem of m-best arm identification and introduced a new version of the SR algorithm, called Successive Accepts and Rejects (SAR), that is able to return the set of m-best arms with high probability.

• **The multi-bandit setting and its use in the CBPI algorithms:** Our motivation for studying the best arm identification problem has been its possible application in the rollouts allocation problem in the CBPI algorithms (see Section 1.3.2). Typically at each iteration of CBPI, one uses an exploration phase with a limited budget B in order to identify the best action (the greedy action) for each state in the rollout set. Let us first consider one state s from the rollout set and denote by π the policy at the current iteration. Each action a can be considered as an arm and sampling an arm corresponds to performing a rollout with the initial state-action pair (s, a) and following π. The value obtained from the sampling is the return $R^\pi(s, a)$. Moreover, from the MDP assumption, these returns are i.i.d. with expected value $Q^\pi(s, a)$. Therefore, identifying the greedy action corresponds to identifying the arm with the highest action-value function. As a consequence, the problem of allocating the rollouts can be cast perfectly as a best arm identification problem. However, in the rollout allocation problem, the objective is to identify the greedy action simultaneously for all of the states in the rollout set. Thus, as several bandit problems must be solved in parallel, this gives rise to a new problem, called multi-bandit best arm identification. Our objective is to study this new problem under the fixed budget setting. We first argue that trivial extensions of the existing algorithms for best arm identification are not satisfactory for this problem. One may first think of extending the Hoeffding Races, an algorithm designed for the single-bandit with fixed confidence setting, to the multi-bandit with fixed budget setting. Nevertheless, as shown by Audibert et al. (2010) and more recently by Kaufmann and Kalyanakrishnan (2013), this class of algorithms is not efficient in practice neither in the fixed budget nor in the fixed confidence settings. Note that, a very likely reason for this is the fact that Hoeffding Races sample uniformly over the arms that have not been discarded yet. Second, another idea to address the multi-bandit setting is a naive application of UCB-E in the multi-bandit setting that would pull at each time step the arm with the highest upper bound over all of the arms from all of the bandits. Therefore this algorithm would only identify the best arm among all the arms of all the bandits and not the best arm of each bandit.

In this dissertation, we are interested in designing new algorithms, for the multi-bandit setting (we also tackle their (m,ϵ) version). The proposed algorithms, which we call GapE, are extensions of the UCB-E algorithm. Roughly speaking, GapE algorithms pull the arm that has the smallest gap, i.e. the smallest difference between its mean and the mean of the best arm within the same bandit. The hope is that relying on the gap, a notion that is relative to each bandit, will permit to balance the error simultaneously in all the bandits at the same time. This hope is confirmed by an analysis providing a bound, the first provided for an algorithm in the multi-bandit setting, on the probability of error of GapE algorithms after n pulls. Even though, in a fashion similar to UCB-E, GapE algorithms assume the knowledge of the complexity of the problem, we show that their adaptive version (when the complexity is estimated as we progress) behave well in practice. Moreover, GapE algorithms compare well to the previously proposed multi-bandit methods (e.g., Deng et al. 2011) as well as to those introduced subsequently to our work (e.g., Wang et al. 2013). More specifically, Deng et al. (2011) proposed an

ϵ-greedy heuristic to the multi-bandit problem with no associated analysis, and Wang et al. (2013) showed that the SAR algorithm, mentioned above, could be extended, with provable guarantees, to the problem of simultaneously identifying the m-best arm(s) in each of the M bandits problems.

Chapter 3
Classification-based Policy Iteration with a Critic

In this chapter,[1] we give our first variant of classification-based policy iteration (CBPI) algorithms. We study the effect of adding a value function approximation component (critic) to CBPI algorithms. The idea is to use a critic to approximate the return after we truncate the rollout trajectories. This allows us to control the bias and variance of the rollout estimates of the action-value function. Therefore, the introduction of a critic can improve the accuracy of the rollout estimates, and as a result, enhance the performance of the CBPI algorithm. We present a new CBPI algorithm, called *direct policy iteration with critic* (DPI-Critic), and provide its finite-sample analysis when the critic is based **1)** on the least-squares policy iteration (LSTD) method, and **2)** on the Bellman residual minimization (BRM) method. We empirically evaluate the performance of DPI-Critic and compare it with direct policy iteration (DPI) introduced in Chapter 2 (Section 1.3.2) and least-squares policy iteration (LSPI) in two benchmark reinforcement learning problems.

Contents

1 Introduction

As discussed in Section 1.3.2, at each iteration, given the current policy π, classification-based policy iteration algorithms: **1)** replaces the policy evaluation step (approximating the action-value function over the entire state-action space) with computing rollout estimates of Q^π over a finite number of states $\mathcal{D} = \{x_i\}_{i=1}^{N'}$, called the *rollout set*, and

[1]This chapter is an extended version of our ICML paper (Gabillon et al., 2011b).

the entire action space, and **2)** casts the policy improvement step as a *classification* problem to find a policy in a given hypothesis space that best predicts a greedy action at every state (see e.g., Lagoudakis and Parr 2003b, Fern et al. 2006, Lazaric et al. 2010a).

As it is suggested by both theoretical and empirical analysis, the performance of the classification-based approximate policy iteration (API) algorithms is closely related to the accuracy in estimating a greedy action at each state of the rollout set, which itself depends on the accuracy of the rollout estimates of the action-values. Thus, it is quite important to balance the bias and variance of the rollout estimates, $\widehat{Q}^\pi$'s, that depend on the length m of the rollout trajectories. While the bias in $\widehat{Q}^\pi$ (i.e., the difference between $\widehat{Q}^\pi$ and the actual Q^π) decreases as m becomes larger, its variance (due to the stochasticity in the MDP transitions and rewards) increases with the value of m. Although the bias and variance of $\widehat{Q}^\pi$ estimates may be optimized by the value of m, when the *budget*, i.e., the number of calls to the generative model, is limited, it may not be possible to find an m that guarantees an accurate enough training set. A particular example of this is the case of goal-based problems where a non-zero reward is only obtained when a terminal state is reached. In this case, truncating the rollouts leads to uninformative rollout estimates and prevents from learning.

A possible approach to address this bias/variance problem is to introduce a *critic* that provides an approximation of the value function. In this approach, we define each $\widehat{Q}^\pi$ estimate as the average of the values returned by rollouts of size m plus the critic's prediction of the return from the time step m on. This allows us to use small values of m, thus having a small estimation variance, and at the same time, to rely on the value function approximation provided by the critic to control the bias. The idea is similar to the actor-critic methods (Barto et al., 1983) in which the variance of the gradient estimate in the actor is reduced using the critic's prediction of the value function.

In this chapter, we introduce a new classification-based API algorithm, called *DPI-Critic*, by adding a critic to the *direct policy iteration* (DPI) algorithm (Lazaric et al., 2010b). We provide finite-sample analysis for our proposed algorithm, DPI-Critic, when the critic approximates the value function using **1)** least-squares temporal-difference (LSTD) learning algorithm (Bradtke and Barto, 1996), or **2)** the Bellman residual minimization (BRM) (Baird, 1995, Schweitzer and Seidman, 1985). We empirically evaluate the performance of DPI-Critic and compare it with DPI and LSPI (Lagoudakis and Parr, 2003a) on two benchmark reinforcement learning (RL) problems: mountain car and inverted pendulum. The results indicate that DPI-Critic can take advantage of both its components and improve over DPI and LSPI.

2 The DPI-Critic Algorithm

In this section, we outline the algorithm we propose in this chapter, called Direct Policy Iteration with a Critic (DPI-Critic), which is an extension of the DPI algorithm (Lazaric et al., 2010b) by adding a critic. As illustrated in Figure 3.1, DPI-Critic starts with an arbitrary initial policy $\pi_0 \in \Pi$. At each iteration k, we build a set of N

```
**Input:** policy space $\Pi$, state distribution $\mu$
**Initialize:** Let $\pi_0 \in \Pi$ be an arbitrary policy
**for** $k = 0, 1, 2, \dots$ **do**
  Construct the rollout set $\mathcal{D}'_k = \{s^{(i)}\}_{i=1}^{N'}$, $s^{(i)} \overset{\text{iid}}{\sim} \mu$
  • *Critic:*
  Construct the set $\mathcal{D}_k$ of $N$ samples (e.g., by following a trajectory or by using the generative model)
  $\widehat{v}^{\pi_k} \leftarrow$ VF-APPROX($\mathcal{D}_k$)        **(critic)**
  • *Rollout:*
  **for all** states $s^{(i)} \in \mathcal{D}'_k$ and actions $a \in \mathcal{A}$ **do**
    **for** $j = 1$ to $M$ **do**
      Perform a rollout and return $R_j(s^{(i)}, a)$
    **end for**
    $\widehat{Q}^{\pi_k}(s^{(i)}, a) = \frac{1}{M}\sum_{j=1}^{M} R_j(s^{(i)}, a)$
  **end for**
  $\pi_{k+1} \in \text{argmin}_{\pi\in\Pi} \widehat{\mathcal{L}}_k^{\Pi}(\widehat{\mu}; \pi)$        **(classifier)**
**end for**
```

Figure 3.1: The pseudo-code of the DPI-Critic algorithm.

samples $\mathcal{D}_k$, called the *critic training set*. The critic uses $\mathcal{D}_k$ in order to compute $\widehat{v}^{\pi_k}$, an approximation of the value function of the current policy π_k. Then, a new policy π_{k+1} is computed from π_k, as the best approximation of the greedy policy with respect to Q^{π_k}, by solving a cost-sensitive classification problem. Similarly to DPI, DPI-Critic is based on the *loss function* $\epsilon'_{\pi_k}(\cdot; \pi)$ and *expected error* $\mathcal{L}^{\Pi}_{\pi_k}(\mu; \pi)$ that are defined as

$$\forall s \in \mathcal{S},\ \ \epsilon'_{\pi_k}(s; \pi) = \max_{a\in\mathcal{A}} Q^{\pi_k}(s, a) - Q^{\pi_k}\Big(s, \pi(s)\Big), \qquad \mathcal{L}^{\Pi}_{\pi_k}(\mu; \pi) = \int_{\mathcal{S}} \epsilon'_{\pi_k}(s; \pi)\mu(ds)\ .$$

To simplify the notation we use $\mathcal{L}^{\Pi}_k$ instead of $\mathcal{L}^{\Pi}_{\pi_k}$ and ϵ'_k instead of ϵ'_{π_k}. In order to minimize this loss, a *rollout set* $\mathcal{D}'_k$ is built by sampling N' states i.i.d. from a distribution μ. For each state $s^{(i)} \in \mathcal{D}'_k$ and each action $a \in \mathcal{A}$, M independent estimates $\{R^{\pi_k}_j(s^{(i)}, a)\}_{j=1}^{M}$ are computed, where

$$R^{\pi_k}_j(s^{(i)}, a) = R^{\pi_k,m}_j(s^{(i)}, a) + \gamma^m \widehat{v}^{\pi_k}(s^{(i,j)}_m)\ , \tag{3.1}$$

in which $R^{\pi_k,m}_j(s^{(i)}, a)$ is the outcome of a rollout of size m, i.e.,

$$R^{\pi_k,m}_j(s^{(i)}, a) = r(s^{(i)}, a) + \sum_{t=1}^{m-1} \gamma^t r(s^{(i,j)}_t, \pi_k(s^{(i,j)}_t))\ , \tag{3.2}$$

and $\widehat{v}^{\pi_k}(s^{(i,j)}_m)$ is the critic's estimate of the value function at state $s^{(i,j)}_m$. In Equation 3.2, $(s^{(i)}, s^{(i,j)}_1, s^{(i,j)}_2, \dots, s^{(i,j)}_m)$ is the trajectory induced by taking action a at state $s^{(i)}$ and following the policy π_k afterwards, i.e., $s^{(i,j)}_1 \sim p(\cdot|s^{(i)}, a)$ and $s^{(i,j)}_t \sim p\Big(\cdot|s^{(i,j)}_{t-1}, \pi_k(s^{(i,j)}_{t-1})\Big)$ for $t \geq 2$. An estimate of the action-value function of the policy π_k is then obtained by averaging the M estimates as

$$\widehat{Q}^{\pi_k}(s^{(i)}, a) = \frac{1}{M}\sum_{j=1}^{M} R_j^{\pi_k}(s^{(i)}, a) \ . \tag{3.3}$$

Given the action-value function estimates, the *empirical loss* and *empirical error* are defined as

$$\forall s \in \mathcal{S}, \ \ \widehat{\epsilon}'_{\pi_k}(s;\pi) = \max_{a\in\mathcal{A}} \widehat{Q}^{\pi_k}(s,a) - \widehat{Q}^{\pi_k}\Big(s,\pi(s)\Big), \qquad \widehat{\mathcal{L}}_k^{\Pi}(\widehat{\mu};\pi) \ = \frac{1}{N'}\sum_{i=1}^{N'} \widehat{\epsilon}'_{\pi_k}(s;\pi). \tag{3.4}$$

Finally, DPI-Critic makes use of a classifier which solves a cost-sensitive classification problem and returns a policy that minimizes the empirical error $\widehat{\mathcal{L}}_k^{\Pi}(\widehat{\mu};\pi)$ over the policy space Π.

As it can be seen from Equation 3.1, the main difference between DPI-Critic and DPI is that after m steps DPI rollouts are truncated and the return thereafter is implicitly set to 0, while in DPI-Critic an approximation of the value function learned by the critic is used to predict this return. Hence, with a fixed rollout size m, even if the critic is a rough approximation of the value function, whenever its accuracy is higher than the implicit prediction of 0 in DPI, the rollouts in DPI-Critic are expected to be more accurate than those in DPI. Similarly, we expect DPI-Critic to obtain the same accuracy as DPI with a smaller rollout size, and as a result, a smaller number of interactions with the generative model. In fact, while in DPI decreasing m leads to a smaller variance and a larger bias, in DPI-Critic the increase in the bias is controlled by the critic. Finally, it is worth noting that DPI-Critic still benefits from the advantages of the classification-based approach to policy iteration compared to value-function-based API algorithms such as LSPI. This is due to the fact that DPI-Critic still relies on approximating the policy improvement step, and thus similar to DPI, whenever approximating good policies is easier than their value functions, DPI-Critic is expected to perform better than its value-function-based counterparts. Furthermore, while DPI-Critic only needs a rough approximation of the value function at certain states, value-function-based API methods, like LSPI, need an accurate approximation of the action-value function over the entire state-action space, and thus, they usually require more samples than the critic in DPI-Critic.

In DPI-Critic, the critic may use any value function approximation method in order to compute $\widehat{v}^{\pi_k}$. However, in this chapter, we first consider critics based on LSTD and report the theoretical analysis of DPI-Critic with BRM in Appendix B.

3 Error Propagation

In this section, we first show how the expected error is propagated through the iterations of DPI-Critic. We then analyze the error between the value function of the policy obtained by DPI-Critic after K iterations and the optimal value function in η-norm, where η is a distribution over the states which might be different from the sampling distribution μ. Let P_π be the transition kernel for policy π, i.e., $P_\pi(dy|s) = p\big(dy|s,\pi(s)\big)$. It defines two related operators: a right-linear operator, $P_\pi\cdot$, which maps any $v \in \mathcal{B}^v(\mathcal{S};Q_{\max})$ to $(P_\pi v)(s) = \int v(y)P_\pi(dy|s)$, and a left-linear operator, $\cdot P_\pi$, that returns $(\eta P_\pi)(dy) = \int P_\pi(dy|s)\eta(ds)$ for any distribution η over $\mathcal{S}$.

From the definitions of ϵ'_{π_k}, T_π, and T, we have $\epsilon'_{\pi_k}(\pi_{k+1}) = Tv^{\pi_k} - T_{\pi_{k+1}}v^{\pi_k}$. We define the operator $E_k = (I-\gamma P_{\pi_{k+1}})^{-1}$, which is well-defined since $P_{\pi_{k+1}}$ is a stochastic kernel and $\gamma < 1$. We are interested in finding an upper bound on the **loss** $l_k \triangleq v^* - v^{\pi_k}$. We deduce the following pointwise inequalities:

Lemma 3.1. *After K iterations of DPI-Critic we have*

$$l_K = v^* - v^{\pi_K} \leq (\gamma P_*)^K(v^* - v^{\pi_0}) + \sum_{k=0}^{K-1}(\gamma P_*)^{K-k-1}E_k\epsilon'_{\pi_k}(\pi_{k+1}). \tag{3.5}$$

Proof. See Appendix A.1. □

Equation 3.5 shows how the error at each iteration k of DPI-Critic, $\epsilon'_{\pi_k}(\pi_{k+1})$, is propagated through the iterations and appears in the final error of the algorithm, $v^* - v^{\pi_K}$. Since we are interested in bounding the final error in η-norm, which might be different than the sampling distribution μ, we use one of the following assumptions:

Assumption 1. *For any policy $\pi \in \Pi$ and any non-negative integers t_1 and t_2, there exists a constant $C_{\eta,\mu}(t_1,t_2) < \infty$ such that $\eta(P_*)^{t}_1(P_\pi)^{t}_2 \leq C_{\eta,\mu}(t_1,t_2)\mu$. We define $C_{\eta,\mu} = (1-\gamma)^2\sum_{t_1=0}^{\infty}\sum_{t_2=0}^{\infty}\gamma^{t_1+t_2}C_{\eta,\mu}(t_1,t_2)$.*

Assumption 2. *For any $s \in \mathcal{S}$ and any $a \in \mathcal{A}$, there exist a constant $C_\mu < \infty$ such that $p(\cdot|s,a) \leq C_\mu\mu(\cdot)$.*

Note that *concentrability coefficients* similar to $C_{\eta,\mu}$ and C_μ were previously used in the ℓ_p-analysis of fitted value iteration (Munos, 2007, Munos and Szepesvári, 2008) and approximate policy iteration (Antos et al., 2008). We now state our main result.

Theorem 3.1. *Let Π be a policy space with finite VC-dimension h and π_K be the policy generated by DPI-Critic after K iterations. Let M be the number of rollouts per state-action and N' be the number of samples drawn i.i.d. from a distribution μ over $\mathcal{S}$ at each iteration of DPI-Critic. Then, we have*

$$||l_k||_{1,\eta} \leq \frac{1}{1-\gamma}\left[\frac{1}{(1-\gamma)}C_{\eta,\mu}||\epsilon'_{\pi_k}(\pi_{k+1})||_{1,\mu} + 2\gamma^K R_{\max}\right], \qquad \textit{under Assumption 1}$$

$$||l_k||_{\infty} \leq \frac{1}{1-\gamma}\left[\frac{1}{(1-\gamma)}C_{\mu}||\epsilon'_{\pi_k}(\pi_{k+1})||_{1,\mu} + 2\gamma^K R_{\max}\right], \qquad \textit{under Assumption 2}$$

Proof. See Appendix A.1. □

4 Finite-Sample Analysis

In this section, we provide a finite-sample analysis and give a bound on the error incurred at each iteration of DPI-Critic, ϵ'_{π_k}, that appears in the error propagation analysis (see Theorem 3.1). We study DPI-Critic when the critic uses pathwise-LSTD (Lazaric et al., 2010c, 2012). In practice, we may use any form of the LSTD algorithm, however, we restrict ourselves to pathwise-LSTD because it is the only LSTD algorithm for which there exists a finite-sample analysis. In order to use the existing finite-sample bounds for pathwise-LSTD, we introduce the following assumptions.

Assumption 3. *At each iteration k of DPI-Critic, the critic uses a linear function space $\mathcal{F}$ spanned by d bounded basis functions (see Section 1.3.1 in Chapter 1). A data-set $\mathcal{D}_k = \{(S_i, R_i)\}_{i=1}^n$ is built, where S_i's are obtained by following a single trajectory generated by a stationary β-mixing process with parameters $\hat{\beta}, b, \kappa$, and a stationary distribution σ_k equal to the stationary distribution of the Markov chain induced by policy π_k, and $R_i = r\big(S_i, \pi_k(S_i)\big)$.*

Assumption 4. *The rollout set sampling distribution μ is such that for any policy $\pi \in \Pi$ and any action $a \in \mathcal{A}$, $\rho = \mu P_a (P_\pi)^{m-1} \leq C\sigma$, where $C < \infty$ is a constant and σ is the stationary distribution of π. The distribution ρ is the distribution induced by starting at a state sampled from μ, taking action a, and then following policy π for $m-1$ steps.*

Before stating the main results of this section, namely Lemma 3.2 and Theorem 3.2, we report the performance bound for pathwise-LSTD as in Lazaric et al. (2012). Since all of the following statements are true for any iteration k, in order to simplify the notation, we drop the dependency of the variables on k.

We may use different function spaces $\mathcal{F}$ (linear or non-linear) to approximate the value function. Here we consider a linear architecture with parameters $\alpha \in \mathbb{R}^d$ and bounded (by L) basis functions $\{\varphi_j\}_{j=1}^d$, $\|\varphi_j\|_\infty \leq L$ as detailed in Section 1.3.1 of Chapter 2. We denote by $\phi : \mathcal{X} \to \mathbb{R}^d$, $\phi(\cdot) = \big(\varphi_1(\cdot), \ldots, \varphi_d(\cdot)\big)^\top$ the feature vector, and by $\mathcal{F}$ the linear function space spanned by the features φ_j, i.e., $\mathcal{F} = \{f_\alpha(\cdot) = \phi(\cdot)^\top \alpha : \alpha \in \mathbb{R}^d\}$. Now if we define v_k as the truncation (by $V_{\max}$) of the solution of the above linear regression problem, we may bound the *evaluation step error* ϵ_k using the following lemma.

Proposition 3.1 (Theorem 5 in Lazaric et al. 2012). *Let N be the number of samples collected in $\mathcal{D}$ as in Assumption 3 and $\widehat{v}^\pi$ be the approximation of the value function of policy π returned by pathwise-LSTD truncated in the range $[-Q_{\max}, Q_{\max}]$. Then for any $\delta > 0$, we have*

$$||v^\pi - \widehat{v}^\pi||_{2,\sigma} \leq e_{LSTD} = \left[\frac{2}{\sqrt{1-\gamma^2}}\Big(2\sqrt{2}\inf_{f\in\mathcal{F}}||v^\pi - f||_{2,\sigma} + \mathsf{e}_2\Big) + \frac{2}{1-\gamma}\mathsf{e}_3 + \mathsf{e}_1\right]$$

with probability $1-\delta$ (with respect to the samples in $\mathcal{D}$), where

(1) $\mathsf{e}_1 = 24Q_{\max}\sqrt{\frac{2\Lambda_1(N,d,\delta/4)}{N}}\max\{\frac{\Lambda_1(N,d,\delta/4)}{b},1\}^{1/\kappa}$, *in which* $\Lambda_1(N,d,\delta) = 2(d+1)\log N + \log\frac{4e}{\delta} + \log^+(\max\{18(6e)^{2(d+1)},\hat{\beta}\})$,
(2) $\mathsf{e}_2 = 12(Q_{\max} + L||\alpha^*||)\sqrt{\frac{2\Lambda_2(N,\delta/4)}{N}}\max\{\frac{\Lambda_2(N,\delta/4)}{b},1\}^{1/\kappa}$, *in which* $\Lambda_2(N,\delta) = \log\frac{4e}{\delta} + \log(\max\{6, N\hat{\beta}\})$ *and* $\alpha^* \in \operatorname{argmin}_{\alpha\in\mathbb{R}^d} ||v^{\pi_k} - f_{\alpha^*}||_{2,\sigma}$,
(3) *$\omega > 0$ is the smallest strictly positive eigenvalue of the Gram matrix with respect to the distribution σ,*
(4) $\mathsf{e}_3 = \gamma Q_{\max}L\sqrt{\frac{d}{\omega}}\Big(\sqrt{\frac{8\log(8d/\delta)}{N}} + \frac{1}{N}\Big)$.

In the following lemma, we derive a bound for the difference between the actual action-value function of policy π and its estimate computed by DPI-Critic.

Lemma 3.2. *Let Assumptions 3 and 4 hold and $\mathcal{D}' = \{s^{(i)}\}_{i=1}^{N'}$ be the rollout set with $s^{(i)} \overset{iid}{\sim} \mu$. Let Q^π be the true action-value function of policy π and $\widehat{Q}^\pi$ be its estimate computed by DPI-Critic using M rollouts of size m (Equations 3.1–3.3). Then for any $\delta > 0$*

$$\max_{a\in\mathcal{A}}\left|\frac{1}{N'}\sum_{i=1}^{N'}\Big[Q^\pi(s^{(i)},a) - \widehat{Q}^\pi(s^{(i)},a)\Big]\right| \le e_1' + e_2' + e_3 + e_4,$$

with probability $1-\delta$ (with respect to the rollout estimates and the samples in the critic training set $\mathcal{D}$), where

$$e_1' = (1-\gamma^m)Q_{\max}\sqrt{\frac{2\log(4|\mathcal{A}|/\delta)}{MN'}}, \qquad e_2' = \gamma^m Q_{\max}\sqrt{\frac{2\log(4|\mathcal{A}|/\delta)}{MN'}},$$

$$e_3 = 24\gamma^m Q_{\max}\sqrt{\frac{2\Lambda(N',d,\frac{\delta}{4|\mathcal{A}|M})}{N'}}, \qquad e_4 = 2\gamma^m\sqrt{C}\, e_{LSTD}\ ,$$

with $\Lambda(N',d,\delta) = \log\Big(\frac{9e}{\delta}(12N'e)^{2(d+1)}\Big)$.

Proof. We prove the following series of inequalities:

$$\begin{aligned}
&\left|\frac{1}{N'}\sum_{i=1}^{N'}\Big[Q^\pi(s^{(i)},a) - \widehat{Q}^\pi(s^{(i)},a)\Big]\right| \\
&\overset{(a)}{=} \left|\frac{1}{MN'}\sum_{i=1}^{N'}\sum_{j=1}^{M}\Big[Q^\pi(s^{(i)},a) - R_j^\pi(s^{(i)},a)\Big]\right| \\
&\overset{(b)}{\le} \left|\frac{1}{MN'}\sum_{i=1}^{N'}\sum_{j=1}^{M}\Big[Q_m^\pi(s^{(i)},a) - R_j^{\pi,m}(s^{(i)},a)\Big]\right| + \left|\frac{\gamma^m}{MN'}\sum_{i=1}^{N'}\sum_{j=1}^{M}\Big[\widehat{v}^\pi(s_m^{(i,j)}) - \mathbb{E}_{s\sim\nu_i}[v^\pi(s)]\Big]\right| \\
&\overset{(c)}{\le} e_1' + \left|\frac{\gamma^m}{MN'}\sum_{i=1}^{N'}\sum_{j=1}^{M}\Big[\widehat{v}^\pi(s_m^{(i,j)}) - v^\pi(s_m^{(i,j)})\Big]\right| + \left|\frac{\gamma^m}{MN'}\sum_{i=1}^{N'}\sum_{j=1}^{M}\Big[v^\pi(s_m^{(i,j)}) - \mathbb{E}_{s\sim\nu_i}[v^\pi(s)]\Big]\right| \\
&\hspace{30em}\text{w.p. } 1-\delta'
\end{aligned}$$

$$\overset{(d)}{\leq} e'_1 + e'_2 + \frac{\gamma^m}{M}\sum_{j=1}^{M} ||v^\pi - \widehat{v}^\pi||_{1,\widehat{\rho}_j} \qquad \text{w.p. } 1-2\delta'$$

$$\overset{(e)}{\leq} e'_1 + e'_2 + \frac{\gamma^m}{M}\sum_{j=1}^{M} ||v^\pi - \widehat{v}^\pi||_{2,\widehat{\rho}_j} \qquad \text{w.p. } 1-2\delta'$$

$$\overset{(f)}{\leq} e'_1 + e'_2 + e_3 + 2\gamma^m ||v^\pi - \widehat{v}^\pi||_{2,\rho} \qquad \text{w.p. } 1-3\delta'$$

$$\overset{(g)}{\leq} e'_1 + e'_2 + e_3 + 2\gamma^m \sqrt{C} ||v^\pi - \widehat{v}^\pi||_{2,\sigma}$$

$$\overset{(h)}{\leq} e'_1 + e'_2 + e_3 + 2\gamma^m \sqrt{C}\ \epsilon_{LSTD} \qquad \text{w.p. } 1-4\delta'$$

The statement of the lemma is obtained by setting $\delta' = \delta/4$ and taking a union bound over actions.
(a) We use Equation 3.3 to replace $\widehat{Q}^\pi(s^{(i)}, a)$.
(b) We replace $R_j^\pi(s^{(i)}, a)$ from Equation 3.1 and use the fact that $Q^\pi(s^{(i)}, a) = Q_m^\pi(s^{(i)}, a) + \gamma^m \mathbb{E}_{s\sim\nu_i}[v^\pi(s)]$, where $Q_m^\pi(s^{(i)}, a) = \mathbb{E}\Big[r(s^{(i)}, a) + \sum_{t=1}^{m-1}\gamma^t r\Big(s_t^{(i)}, \pi(s_t^{(i)})\Big)\Big]$ and $\nu_i = \delta(s^{(i)})P_a(P_\pi)^{m-1}$, with $\delta(\cdot)$ the Dirac function, is the distribution over states induced by starting at state $s^{(i)}$, taking action a, and then following the policy π for $m-1$ steps. We split the sum using the triangle inequality.
(c) Using the Chernoff-Hoeffding inequality, with probability $1-\delta'$ (with respect to the samples used to build the rollout estimates), we have

$$\left|\frac{1}{MN'}\sum_{i=1}^{N'}\sum_{j=1}^{M}\Big[Q_m^\pi(s^{(i)}, a) - R_j^{\pi,m}(s^{(i)}, a)\Big]\right| \leq e'_1 = (1-\gamma^m)Q_{\max}\sqrt{\frac{2\log(1/\delta')}{MN'}}\ .$$

(d) Using the Chernoff-Hoeffding inequality, with probability $1-\delta'$ (with respect to the last state reached by the rollout trajectories), we have

$$\left|\frac{\gamma^m}{MN'}\sum_{i=1}^{N'}\sum_{j=1}^{M}\Big[v^\pi(s_m^{(i,j)}) - \mathbb{E}_{s\sim\nu_i}[v^\pi(s)]\Big]\right| \leq e'_2 = \gamma^m Q_{\max}\sqrt{\frac{2\log(1/\delta')}{MN'}}\ .$$

We also use the definition of empirical ℓ_1-norm and replace the second term with $||v^\pi - \widehat{v}^\pi||_{1,\widehat{\rho}_j}$, where $\widehat{\rho}_j$ is the empirical distribution corresponding to the distribution $\rho = \mu P_a(P_\pi)^{m-1}$. In fact for any $1 \leq j \leq M$, samples $s_m^{(i,j)}$ are i.i.d. from ρ.
(e) We move from ℓ_1-norm to ℓ_2-norm using the Cauchy-Schwarz inequality.
(f) Note that $\widehat{v}$ is a random variable independent from the samples used to build the rollout estimates. Using Corollary 12 in Lazaric et al. (2012), we have

$$||v^\pi - \widehat{v}^\pi||_{2,\widehat{\rho}_j} \leq 2||v^\pi - \widehat{v}^\pi||_{2,\rho} + e_3(\delta'')$$

with probability $1-\delta''$ (with respect to the samples in $\widehat{\rho}_j$) for any j, and $e_3(\delta'') = 24Q_{\max}\sqrt{\frac{2\Lambda(N',d,\delta'')}{N'}}$. By taking a union bound over all j's and setting $\delta'' = \delta'/M$, we obtain the definition of e_3 in the final statement.
(g) Using Assumption 4, we have $||v^\pi - \widehat{v}||_{2,\rho} \leq \sqrt{C}||v^\pi - \widehat{v}||_{2,\sigma}$.
(h) We replace $||v^\pi - \widehat{v}||_{2,\sigma}$ using Proposition 3.1. □

Using the result of Lemma 3.2, we now prove a performance bound for a single iteration of DPI-Critic.

Theorem 3.2. *Let Π be a policy space with finite VC-dimension $h = VC(\Pi) < \infty$ and μ be a distribution over the state space $\mathcal{S}$. Let N' be the number of states in $\mathcal{D}'_k$ drawn i.i.d. from μ, m be the size of the rollouts, M be the number of rollouts per state-action pair, and $\widehat{v}^{\pi_k}$ be the estimation of the value function returned by the critic. Let Assumptions 3 and 4 hold and $\pi_{k+1} \in \operatorname{argmin}_{\pi\in\Pi} \widehat{\mathcal{L}}^{\Pi}_k(\widehat{\mu};\pi)$ be the policy computed at the k'th iteration of DPI-Critic. Then, for any $\delta > 0$, we have*

$$||\epsilon'_k(\pi_{k+1})||_{1,\mu} = \mathcal{L}^{\Pi}_k(\mu;\pi_{k+1}) \leq \inf_{\pi\in\Pi} \mathcal{L}^{\Pi}_k(\mu;\pi) + 2(e'_0 + e'_1 + e'_2 + e_3 + e_4), \tag{3.6}$$

with probability $1-\delta$, where

$$e'_0 = 16Q_{\max}\sqrt{\frac{2}{N'}\left(h\log\frac{eN'}{m} + \log\frac{32}{\delta}\right)}\,.$$

The proof follows similar steps as in Lazaric et al. (2010b) and is reported in Appendix A.2. Plugging this result in Theorem 3.1 leads to the final result of this section, the finite sample performance bounds of DPI-Critic.

Theorem 3.3. *Let*

$$d' = \inf_{\pi\in\Pi} \mathcal{L}^{\Pi}_k(\mu;\pi) \qquad \textit{and} \qquad d_m = \inf_{f\in\mathcal{F}} ||v^{\pi} - f||_{2,\sigma}.$$

where $\mathcal{F}$ is the function space used by the critic and Π is the policy space used by DPI-Critic. After K iterations, and with probability $1-\delta$, the expected loss $\mathbb{E}_\mu[l_k] = \|l_k\|_{1,\mu}$ of DPI-Critic satisfy:

$$\begin{aligned}||l_k||_{1,\eta} \leq \frac{1}{1-\gamma}\Bigg[&\frac{1}{1-\gamma}C_{\eta,\mu}\Big(d' + 2(e'_0 + e'_1 + e'_2 + e_3 \\ &+ 2\gamma^m\sqrt{C}\,(\frac{2}{\sqrt{1-\gamma^2}}\big(2\sqrt{2}d_m + \mathsf{e}_2) + \frac{2}{1-\gamma}\mathsf{e}_3 + \mathsf{e}_1)\Big) + 2\gamma^K R_{\max}\Bigg].\end{aligned} \tag{3.7}$$

Remark 3.1. The terms in the bound of Theorem 3.2 are related to the performance at each iteration of DPI-Critic. The first term, $\inf_{\pi\in\Pi}\mathcal{L}^{\Pi}_k(\mu;\pi)$, is the approximation error of the policy space Π, i.e., the best approximation of the greedy policy in Π. Since the classifier relies on a finite number of samples in its training set, it is not able to recover the optimal approximation and incurs an additional estimation error e'_0 which decreases as $O(N'^{-1/2})$. Furthermore, the training set of the classifier is built according to action-value estimates, whose accuracy is bounded by the remaining terms. The term e'_1 accounts for the variance of the rollout estimates due to the limited number of rollouts for each state in the rollout set. While it decreases as M and N' increase, it increases with m, because longer rollouts have a larger variance due to the stochasticity in the MDP dynamics. The terms e'_2, e_3, and e_4 are related to the bias induced by truncating the rollouts. They all share a factor γ^m decaying exponentially with m and are strictly related to the critic's prediction of the return from m on. While e_3 depends on the specific function approximation algorithm used by the critic (LSTD in our analysis) just through the dimension d of the function space $\mathcal{F}$, e_4 is strictly related to LSTD's performance, which depends on the size N of its training set and the accuracy of its function space, i.e., the approximation error $\inf_{f\in\mathcal{F}} ||v^\pi - f||_{2,\sigma}$.

Remark 3.2 (Advantage of using a critic). We now compare the result of Theorem 3.2 with the corresponding result for DPI in Lazaric et al. (2010b), which bounds the performance as

$$\mathcal{L}_k^{\Pi}(\mu;\pi_{k+1}) \leq \inf_{\pi\in\Pi} \mathcal{L}_k^{\Pi}(\mu;\pi) + 2(e'_0 + e'_1 + \gamma^m Q_{\max}). \tag{3.8}$$

While the approximation error $\inf_{\pi\in\Pi} \mathcal{L}_k^{\Pi}(\mu;\pi)$ and the estimation errors e'_0 and e'_1 are the same in Equations 3.6 and 3.8, the difference in the way that these algorithms handle the rollouts after m steps leads to the term $\gamma^m Q_{\max}$ in DPI and the terms e'_2, e_3, and e_4 in DPI-Critic. The terms e'_2, e_3, and e_4 have the term $\gamma^m Q_{\max}$ multiplied by a factor which decreases with the number of rollout states N', the number of rollouts M, and the size of the critic training set N. For large enough values of N' and N, this multiplicative factor is smaller than 1, thus making $e'_2 + e_3 + e_4$ smaller than $\gamma^m Q_{\max}$ in DPI. Furthermore, since these e values upper bound the difference between quantities bounded in $[-Q_{\max}, Q_{\max}]$, their values cannot exceed $\gamma^m Q_{\max}$. This comparison supports the idea that introducing a critic could improve the accuracy of the truncated rollout estimates by reducing the bias with no increase in the variance.

Remark 3.3 (Critic trade-off). Although Theorem 3.2 reveals the potential advantage of DPI-Critic with respect to DPI, the comparison in Remark 3.2 does not take into consideration that DPI-Critic uses N samples more than DPI, thus making the comparison potentially unfair. We now analyze the case where the total budget (number of calls to the generative model) of DPI-Critic is fixed to B, a constraint that we will later use in our experiments. The total budget is split into two parts: **1)** $B_R = B(1-p)$ the budget available for the rollout estimates, and **2)** $B_C = Bp = n$ the number of samples used by the critic, where $p \in (0,1)$ is the *critic ratio* of the total budget. By substituting B_R and B_C in the bound of Theorem 3.2 and setting $M = 1$, we note that for a fixed m, while increasing p increases the estimation error terms e'_0, e'_1, e'_2, and e_3 (the rollout set becomes smaller), it decreases the estimation error of LSTD e_4 (the critic's training set becomes larger). This trade-off (later referred to as the *critic trade-off*) is optimized by a specific value $p = p^*$ which minimizes the expected error of DPI-Critic. By comparing the bounds of DPI and DPI-Critic, we first note that for any fixed p, DPI benefits from a larger number of samples to build the rollout estimates, thus has smaller estimation errors e'_0 and e'_1 with respect to DPI-Critic. However, as pointed out in Remark 3.2, the bias term $\gamma^m Q_{\max}$ in the DPI bound is always worse than the corresponding term in the DPI-Critic bound. As a result, whenever the advantage obtained by relying on the critic is larger than the loss in having a smaller number of rollouts, we expect DPI-Critic to outperform DPI. Whether this is the case depends on a number of factors such as the dimensionality and the approximation error of the space $\mathcal{F}$, the size of the rollouts m, and the size N' of the rollout set.

Remark 3.4 (Reuse of samples). According to Assumption 1 the samples in the critic's training set are completely independent from those used in building the rollout estimates. A more data-efficient version of the algorithm can be devised as follows: We first simulate all the trajectories used in the computation of the rollouts and use the last few transitions of each to build the critic's training set $\mathcal{D}_k$. Then, after the critic (LSTD or BRM) computes an estimate of the value function using the samples in $\mathcal{D}_k$, the action-values of the states in the rollout set $\mathcal{D}'_k$ are estimated as in Equations 3.1–3.3. This way the function approximation step does not change the total budget. We call this version of the algorithm *Combined DPI-Critic* (CDPI-Critic). From a theoretical point of view, the main problem is that the samples in $\mathcal{D}_k$ are no longer drawn from the stationary distribution σ_k of the policy under evaluation π_k. However, the samples in $\mathcal{D}_k$ are collected at the end of the rollout trajectories of length m obtained by following π_k, and thus, they are drawn from the distribution $\rho = \mu P_a(P_{\pi_k})^{m-1}$ that approaches σ_k as m increases. Depending on the mixing rate of the Markov chain induced by π_k, the difference between ρ and σ_k could be relatively small, thus supporting the conjecture that CDPI-Critic may achieve a similar performance to DPI-Critic without the overhead of N independent samples. While we leave a detailed theoretical analysis of CDPI-Critic as future work, we use it in the experiments of Section 5.

Remark 3.5 (New rollout trade-off). The parameter m in DPI-Critic balances the errors in evaluating the value function and the policy. Up to constants and logarithmic factors, the bound in Equation 3.7 has the form

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

The value function approximation error term, $\gamma^m\left(d_m + \sqrt{\frac{1}{N}}\right)$, tends to zero for large values of m. Although this would suggest to have large values for m, when having a fixed budget B, the size of the rollout set $\mathcal{D}'$ would correspondingly decreases as $N' = O(B/m)$ and would also make the influence of the estimation error of the classifier bigger (see e'_0 in Theorem 3.2), thus decreasing the accuracy of the classifier. This leads to a new trade-off, similar to the one discussed in Section 1.3.2 of Chapter 1 for DPI, between long rollouts and the number of states in the rollout sets. However the solution to this trade-off strictly depends on the capacity of the value function space $\mathcal{F}$. A rich value function space would lead to solving the trade-off for small values of m. On the other hand, when the value function space is poor, or when there is no value function as in the case of DPI, m should be selected in a way to guarantee large enough rollout sets (parameters N'), and at the same time, a sufficient number of rollouts (parameter M).

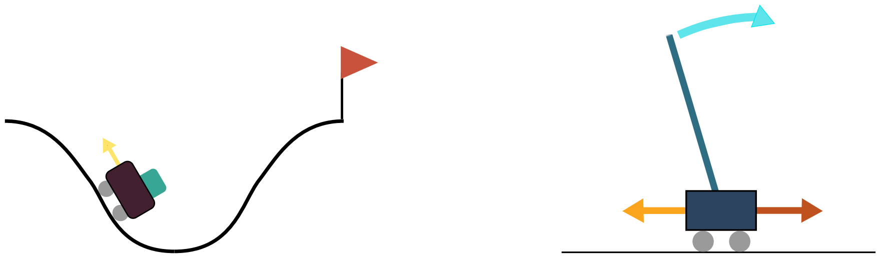


Figure 3.2: ***(Left)*** The Mountain Car (MC) problem in which the car needs to learn to oscillate back and forth in order to build up enough inertia to reach the top of the one-dimensional hill. ***(Right)*** The Inverted pendulum (IP) is the problem of balancing a pendulum at the upright position by applying force to the cart it is attached to.

Remark 3.6 (Recent theoretical analysis). Farahmand et al. (2012) analyze a generic version of CBPI algorithms called CAPI. In CAPI, the policy improvement uses the cost-sensitive loss of Equation 3.4 while the policy evaluation can be any algorithm that returns an approximation $\widehat{Q}^\pi$ of Q^π. CAPI contains therefore the DPI and the DPI-Critic algorithms. Farahmand et al. (2012) provided a finite sample error analysis of CAPI, which allows general policy evaluation algorithms, handles non-parametric policy spaces, and provides a faster convergence rate than existing results. Note that using nonparametric policies extends our analysis which is limited to policy spaces with finite VC dimension. They also provide a new error propagation result for classification-based RL algorithms that shows that the errors at the later iterations play a more important role in the quality of the final policy. Faster rates of convergence than existing results are obtained thanks to the use the notion of local Rademacher complexity. They also benefit from the notion of action-gap regularity of the problem which means that choosing the right action at each state may not require a precise estimate of the action-value function.

5 Experiments

In this section, we report the empirical evaluation of DPI-Critic with LSTD critic and compare it to DPI (built on truncated rollouts) and LSPI (built on value function approximation). In the experiments we show that DPI-Critic, by combining truncated rollouts and function approximation, can improve over DPI and LSPI.

5.1 Setting

We consider two standard goal-based RL problems, namely, *Mountain Car* and *Inverted pendulum*, described below.

- *Mountain Car* (MC) is the problem of driving a car up to the top of a one-dimensional hill (see Figure 3.2). The car is not powerful enough to accelerate directly up the hill, and thus, it must learn to oscillate back and forth to build up enough inertia. There are three possible actions: forward $(+1)$, reverse (-1), and stay (0). The reward is -1 for all the states but the goal state at the top of the hill, where the episode ends with a reward 0. The discount factor is set to $\gamma = 0.99$. Each state s consists of the pair $(x, \dot{x})$ where x is the position of the car and $\dot{x}$ is its velocity. We use the formulation described in Dimitrakakis and Lagoudakis (2008) with uniform noise in $[-1, 1]$ added to the actions.

- *Inverted Pendulum* (IP) is the problem of balancing a pendulum at the upright position by applying force to the cart it is attached to. There are three possible actions: left (-50N), right $(+50\text{N})$, and stay (0N), with uniform noise in $[-10, 10]$ added to them. The reward is 0 as long as the pendulum is above the horizontal line. The episode ends with reward -1 when the pendulum goes below the horizontal line. The discount factor is set to 0.95. We use the formulation described in Lagoudakis and Parr (2003a) and use the same feature space and the same strategy to collect samples.

The value function is approximated using a linear space spanned by a set of radial basis functions (RBFs) evenly distributed over the state space plus a constant offset. More precisely, we uniformly divide the 2-dimensional state space into a number of regions and place a Gaussian function at the center of each of them. We set the standard deviation of the Gaussian functions to the width of a region. The function space to approximate the action-value function in LSPI is obtained by replicating the state-features for each action. The critic training set is built using one-step transitions from states drawn from a uniform distribution over the state space, while LSPI is trained off-policy using samples from a random policy. In the IP problem, we use the same implementation, features, and critic's training set as in Lagoudakis and Parr (2003a) with $\gamma = 0.95$.

In both domains, the function space to approximate the action-value function in LSPI is obtained by replicating the state-features for each action as suggested in Lagoudakis and Parr (2003a). Similarly to Dimitrakakis and Lagoudakis (2008), the policy space Π (classifier) is defined by a multi-layer perceptron with 10 hidden units, and is trained using stochastic gradient descent with a learning rate of 0.5 for 400 iterations.[2] In the experiments, instead of directly solving the cost-sensitive multi-class classification step as in Figure 3.1, we minimize the classification error. In fact, the classification error is an upper-bound on the empirical error defined by Equation 3.4. Finally, the rollout set is sampled uniformly over the state space.

Each DPI-based algorithm is run with the same fixed budget B per iteration. As discussed in Remark 3.4, DPI-Critic splits the budget into a rollout budget $B_R = B(1-p)$ and a critic budget $B_C = Bp$, where $p \in (0, 1)$ is the critic ratio. The rollout budget is divided into M rollouts of length m for each action in $\mathcal{A}$ and each state in the rollout set $\mathcal{D}'$, i.e., $B_R = mMN'|\mathcal{A}|$. In CDPI-Critic the critic training set $\mathcal{D}_k$ is built using all transitions in the rollout trajectories except the first one. LSPI is

[2]We noticed that 25 iterations used by Dimitrakakis and Lagoudakis (2008) is not enough for training the classifier in all cases.

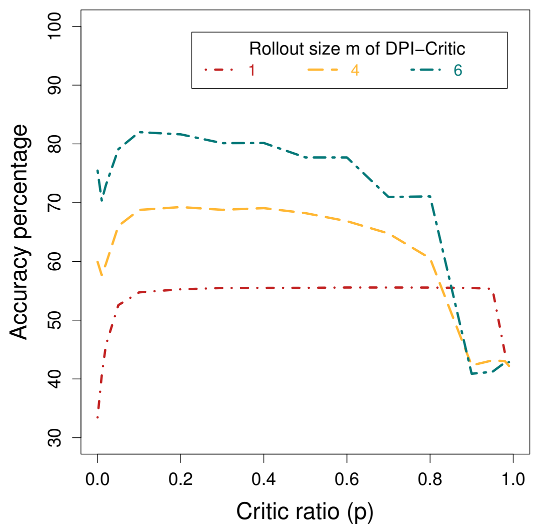


Figure 3.3: The accuracy of training set in the inverted pendulum problem.

run off-policy (i.e., samples are collected once and reused through the iterations) and in order to have a fair comparison, its total number of samples equal to B times the number of iterations (which is 5 in the following experiments).

5.2 Experimental Results

In both MC and IP, the reward function is constant everywhere except at the terminal state. Thus, rollouts are *informative* only if their trajectories reach the terminal state. Although this would suggest to have large values for the rollout size m, the size of the rollout set would correspondingly decrease as $N' = O(B/m)$, leading to a trade-off (referred to as the *rollout trade-off* in Remark 3.6). The solution to this trade-off will strictly depend on the accuracy of the estimate of the return after a rollout is truncated. Prior to considering the final performance of the algorithms, we also carried out a study on the role of parameters M, m, in producing accurate estimates of the action-value function of the states in the rollout set before their usage in the classifier.

5.2.1 Accuracy of the Training Set

Figure 3.3 shows the accuracy of the training set of DPI-Critic with respect to p for different values of m, in the IP domain. At $p = 0$ we report the performance of the DPI algorithm. The accuracy acc is computed as the percentage of the states in the rollout set at which a true greedy action is correctly identified. Let π be a fixed policy. With a rollout set containing N' states acc is computed as:

$$acc = \frac{1}{N'} \sum_{i=1}^{N'} \mathbb{I}\{a_i^* = \hat{a}_i\},$$

where $a_i^* \in \operatorname{argmax}_{a\in\mathcal{A}} Q^\pi(s^{(i)}, a)$ and $\hat{a}_i \in \operatorname{argmax}_{a\in\mathcal{A}} \widehat{Q}^\pi(s^{(i)}, a)$ are an actual greedy action and a greedy action estimated by the algorithm at state $s^{(i)} \in \mathcal{D}'$, respectively.

The budget B is set to 2000, N' is set to 20, and the values of M are computed as

$$M(p,m) = \frac{B}{N'|\mathcal{A}|} \times \frac{1-p}{m} ,$$

where the first term is a constant and the second one indicates that M decreases linearly with p with a coefficient inversely proportional to m. The results are averaged over 1000 runs. The policy π is fixed as follows:

$$\pi(\theta,\dot{\theta}) = \begin{cases} \text{with probability } 0.2, & \text{random action,} \\ \text{with probability } 0.8, & \text{left,} \quad \text{if } \frac{6\theta}{\pi/2} > \dot{\theta} , \\ & \text{right,} \quad \text{otherwise .} \end{cases}$$

This policy has an average performance of 20 steps to balance the pendulum.

For $m = 1$ and $p = 0$, the rollout size is not long enough to collect any informative reward (i.e., reaching a terminal state). Therefore, the accuracy of the training set is almost the same as for a training set in which the greedy action is selected at random (33% as the domain contains 3 actions). For positive values of p, DPI-Critic adds an approximation of the value function to the rollout estimates. The benefit obtained by using the critic increases with the quality of the approximation (i.e., when p increases). At the same time, when p increases, the number of rollouts M for each rollout state decreases, thus increasing the variance of rollout estimates. For $m = 1$, this reduction has no effect on the accuracy of the training set except when M is forced to be 0 for the values of p close to 1. In fact, when $m = 1$, the variance is limited to the variance introduced by the noise in one single transition and even a very small number of rollouts would be enough to obtain accurate estimates of the action values.

The accuracy of DPI ($p = 0$) improves with rollout size $m = 4$ and $m = 6$. For p close to 0, the critic has a poor performance which causes the rollouts to be less accurate than in DPI. On the other hand, when p increases, the value function approximation is sufficiently accurate to make DPI-Critic improve over the accuracy of DPI. However for $p > 0.5$, the accuracy of the training set starts decreasing. Indeed, as m is large, the variance of $\widehat{Q}^\pi$ estimates is bigger than in the case when $m = 1$. Moreover, for large values of m, M is small. Therefore, with high variance and a small number of rollouts M, the $\widehat{Q}^\pi$ estimates are likely to be inaccurate.

These results show that the introduction of a critic improves the accuracy of the training set for a value of p which balances between having a sufficiently accurate approximation of the critic (p large leads to an accurate critic) and having sufficiently many rollouts M (p small leads to rollouts with less variance). This still does not account for the complete critic trade-off since the parameter N' is fixed. Indeed, in order to return a good approximation of the greedy policy, it is essential for the classifier to both have an accurate training set and a large number of states in the training set. The impact of the number of states in the training set and the propagation through iterations of the benefit obtained from the use of critic is studied in the very next section.

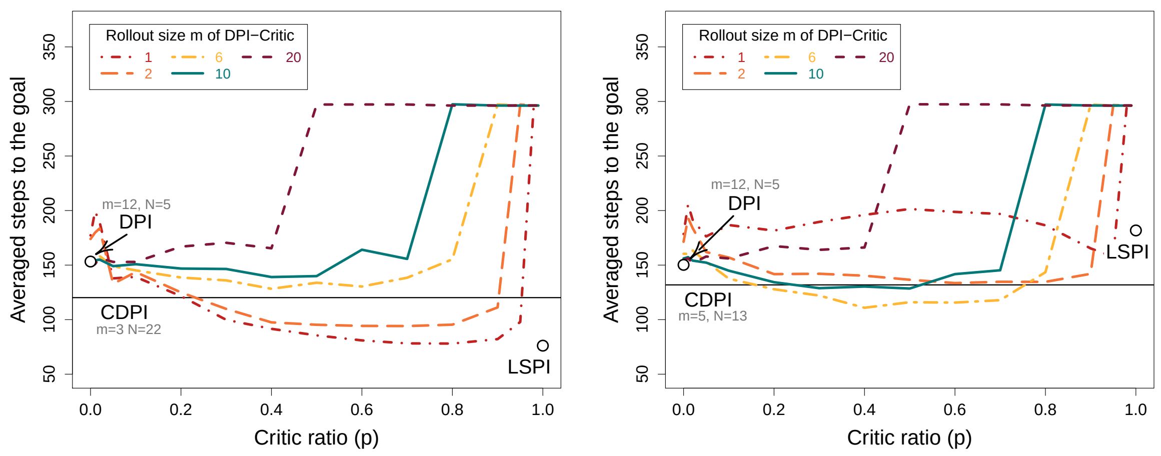


Figure 3.4: Performance of the learned policies in mountain car with a 3×3 RBF grid (left) and a 2×2 RBF grid (right). The total budget B is set to 200. The objective is to minimize the number of steps to the goal.

5.2.2 Comparison of Final Performances

In Figures 3.4 and 3.5, we report the performance of DPI, DPI-Critic, CDPI-Critic, and LSPI. In MC, the performance is evaluated as the number of steps-to-go with a maximum of 300, starting from the bottom of the hill with no initial velocity. In IP, the performance is the number of balancing steps with a maximum of 3000 steps. The performance of each run is computed as the best performance over 5 iterations of policy iteration. The results are averaged over 1000 runs. Although in the graphs we report the performance of DPI and LSPI at $p = 0$ and $p = 1$, respectively, DPI-Critic does not necessarily tend to the same performance as DPI and LSPI when p approaches 0 or 1. In fact, values of p close to 0 correspond to building a critic with very few samples (thus affecting the performance of the critic), while values of p close to 1 correspond to a very small rollout set (thus affecting the performance of the classifier). We tested the performance of DPI and DPI-Critic on a wide range of parameters (m, M, N') but we only report the performance of the best combination for DPI, and show the performance of DPI-Critic for the best choice of M ($M = 1$ was the best choice in all the experiments) and different values of m.

Figure 3.4 shows the learning results in MC with budget $B = 200$. In the left panel, the function space for the critic consists of 9 RBFs distributed over a uniform grid. Such a space is rich enough for LSPI to learn nearly-optimal policies (about 80 steps to reach the goal). On the other hand, DPI achieves a poor performance of about 150 steps, which is obtained by solving the rollout trade-off at $m = 12$ and $N' = 5$. We also report the performance of DPI-Critic for different values of m and p. We note that, as discussed in Remark 3.4, for a fixed m, there exists an optimal value p^* which optimizes the critic trade-off. For very small values of p, the critic has a very small training set and is likely to return a very poor approximation of the return. In this case, DPI-Critic performs similarly to DPI and the rollout trade-off is achieved by $m = 12$, which limits the effect of potentially inaccurate predictions without reducing

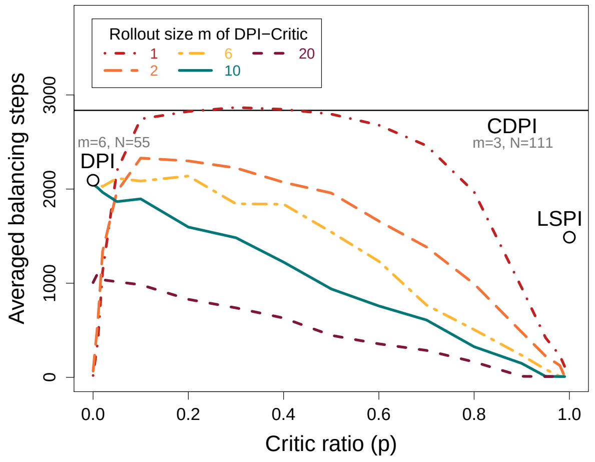


Figure 3.5: Performance of the learned policies in inverted pendulum. The budget is $B = 1000$. The goal is to keep the pendulum balanced with a maximum of 3000 steps.

too much the size of the rollout set. On the other hand, as p increases the accuracy of the critic improves as well, and the best choice for m rapidly reduces to 1, which corresponds to rollouts built almost entirely on the basis of the values returned by the critic. For $m = 1$ and $p \approx 0.8$, DPI-Critic achieves a slightly better performance than LSPI. Finally, the horizontal line represents the performance of CDPI-Critic (for the best choice of m) which improves over DPI without matching the performance of LSPI.

Although this experiment shows that the introduction of a critic in DPI compensates for the truncation of the rollouts and improves their accuracy, most of this advantage is due to the quality of $\mathcal{F}$ in approximating value functions (LSPI itself is nearly-optimal). In this case, the results would suggest the use of LSPI rather than any DPI-based algorithm. In the next experiment, we show that DPI-Critic is able to improve over both DPI and LSPI when $\mathcal{F}$ has a lower accuracy. We define a new space $\mathcal{F}$ spanned by 4 RBFs distributed over a uniform grid. The results are reported in the right panel of Figure 3.4. The performance of LSPI now worsens to 180 steps. Since the quality of the critic returned by LSTD in DPI-Critic is worse than in the case of 9 RBFs, $m = 1$ is no longer the best choice for the rollout trade-off. However, as soon as $p > 0.1$, the accuracy of the critic is still higher than the 0 prediction used in DPI, thus leading to the best size of the rollouts at $m = 6$ (instead of 12 as in DPI), which guarantees a large enough number of informative rollouts. At the same time, other effects might influence the choice of the best size of rollouts m. As it can be noticed, for $m = 6$ and $p \approx 0.5$, DPI-Critic successfully takes advantage of the critic to improve over DPI, and at the same time, it achieves a better performance than LSPI. Unlike LSPI, DPI-Critic computes its action-value estimates by combining informative rollouts and the critic value function, thus obtaining estimates which cannot be represented by the action-value function space used by LSPI. Additionally, similarly to DPI, DPI-Critic performs a policy approximation step which could lead to better policies with respect to those obtained by LSPI.

Finally, Figure 3.5 displays the results of similar experiments in IP with $B = 1000$.

In this case, although the function space is not accurate enough for LSPI to learn good policies, it is helpful in improving the accuracy of the rollouts with respect to DPI. When $p > 0.05$, $m = 1$ is the rollout size which optimizes the rollout trade-off. In fact, since by following a random policy the pendulum falls after very few steps, rollouts of length one still allow to collect samples from the terminal state whenever the starting state is close enough to the horizontal line. Hence, with $m = 1$ action-values are estimated as a mix of both informative rollouts and the critic's prediction, and at the same time, the classifier is trained on a relatively large training set. Finally, it is interesting to note that in this case CDPI-Critic obtains the same nearly-optimal performance as DPI-Critic.

6 Conclusions & Future Work

DPI-Critic adds value function approximation to the classification-based approach to policy iteration, and more specifically to the DPI algorithm (Lazaric et al., 2010a). The motivation behind DPI-Critic is two-fold. **1)** In some settings (e.g., those with delayed reward), DPI action-value estimates suffer from either high variance or high bias (depending on m). Introducing a critic to the computation of the rollouts may significantly reduce the bias with respect to a simple truncation of the rollouts, which in turn allows for shorter rollout size and thus lower variance. **2)** In value-based approaches (e.g., LSPI), it is often difficult to design a function space which accurately approximates action-value functions. In this case, integrating rough approximation of the value function returned by the critic with the rollouts obtained by direct simulation of the generative model may improve the accuracy of the function approximation and lead to better policies.

In Section 4, we theoretically analyzed the performance of DPI-Critic and showed that depending on several factors (notably the function approximation error), DPI-Critic may achieve a better performance than DPI. This analysis is also supported by the experimental results of Section 5, which confirm the capability of DPI-Critic to take advantage of both rollouts and critic, and improve over both DPI and LSPI. Although in some settings either DPI or LSPI might still be the better choice, DPI-Critic seems to be a promising alternative that introduces additional flexibility in the design of the algorithm.

Possible directions for future work include complete theoretical analysis of CDPI-Critic and more detailed comparison of DPI-Critic and LSPI. Another direction is to design new allocation strategies of the rollouts over the states of the rollout set that would improve the current uniform allocation. Indeed this would permit to use, for instance, more rollouts to estimate the action-value of the good actions and to use less rollouts for clear suboptimal actions. Moreover allocating more rollouts on more "important" states or states where the best action is hard to discriminate from the others actions could also be an important target. Motivated by these ideas, in Chapter 5, we study a related problem under the multi-armed bandit setting and design new candidate strategies. More precisely, we discuss in Section 5.1 of Chapter 5 the applications of these new rollout allocation strategies in the classification-based PI framework.

Finally, the next chapter will give a new view on the idea developed in this chapter. Indeed we will see that the introduction of a critic in classification-based PI algorithms possesses close connections with the standard Modified PI algorithm, an algorithm that generalizes the well-known Value Iteration and Policy Iteration algorithms. Moreover, it also serves as an experimental evidence as the usability of classification-based PI algorithms in practice. Specifically, it is shown that these algorithms happen to be the first approximate dynamic programming algorithm to perform well in the game of Tetris.

Appendix

A Proofs of the Main Theorems

In this section we report the proofs of the error propagation (Theorem 3.1) and of the finite sample analysis (Theorem 3.2).

A.1 Error Propagation Proofs

In this section we bound how the errors propagate through the iterations in DPI-Critic.

Proof of Lemma 3.1. From the definitions of ϵ'_{π_k}, T_π, and T, we have $\epsilon'_{\pi_k}(\pi_{k+1}) = Tv^{\pi_k} - T_{\pi_{k+1}}v^{\pi_k}$. We deduce the following pointwise inequalities:

$$\begin{aligned} v^{\pi_k} - v^{\pi_{k+1}} &= T_{\pi_k}v^{\pi_k} - T_{\pi_{k+1}}v^{\pi_k} + T_{\pi_{k+1}}v^{\pi_k} - T_{\pi_{k+1}}v^{\pi_{k+1}} \\ &\le \epsilon'_{\pi_k}(\pi_{k+1}) + \gamma P_{\pi_{k+1}}(v^{\pi_k} - v^{\pi_{k+1}}), \end{aligned}$$

which gives us $v^{\pi_k} - v^{\pi_{k+1}} \le (I - \gamma P_{\pi_{k+1}})^{-1}\epsilon'_{\pi_k}(\pi_{k+1})$. We also have

$$\begin{aligned} v^* - v^{\pi_{k+1}} &= Tv^* - Tv^{\pi_k} + Tv^{\pi_k} - T_{\pi_{k+1}}v^{\pi_k} + T_{\pi_{k+1}}v^{\pi_k} - T_{\pi_{k+1}}v^{\pi_{k+1}} \\ &\le \gamma P_*(v^* - v^{\pi_k}) + \epsilon'_{\pi_k}(\pi_{k+1}) + \gamma P_{\pi_{k+1}}(v^{\pi_k} - v^{\pi_{k+1}}), \end{aligned}$$

which yields

$$\begin{aligned} v^* - v^{\pi_{k+1}} &\le \gamma P_*(v^* - v^{\pi_k}) + \Big[\gamma P_{\pi_{k+1}}(I - \gamma P_{\pi_{k+1}})^{-1} + I\Big]\epsilon'_{\pi_k}(\pi_{k+1}) \\ &= \gamma P_*(v^* - v^{\pi_k}) + (I - \gamma P_{\pi_{k+1}})^{-1}\epsilon'_{\pi_k}(\pi_{k+1}). \end{aligned}$$

Finally, by defining the operator $E_k = (I - \gamma P_{\pi_{k+1}})^{-1}$, which is well-defined since $P_{\pi_{k+1}}$ is a stochastic kernel and $\gamma < 1$, and by induction, we obtain

$$v^* - v^{\pi_K} \le (\gamma P_*)^K(v^* - v^{\pi_0}) + \sum_{k=0}^{K-1}(\gamma P_*)^{K-k-1}E_k\epsilon'_{\pi_k}(\pi_{k+1}).$$

□

Proof of Theorem 3.1. We have $C_{\eta,\mu} \le C_\mu$ for any η. Thus, if the ℓ_1-bound holds for any η, choosing η to be a Dirac at each state implies that the ℓ_∞-bound holds as well. Hence, we only need to prove the ℓ_1-bound. By taking the absolute value point-wise in Equation 3.5 we obtain

$$|v^* - v^{\pi_K}| \le (\gamma P_*)^K|v^* - v^{\pi_0}| + \sum_{k=0}^{K-1}(\gamma P_*)^{K-k-1}(I - \gamma P_{\pi_{k+1}})^{-1}|\epsilon'_{\pi_k}(\pi_{k+1})|.$$

From the fact that $|v^* - v^{\pi_0}| \le \frac{2}{1-\gamma}R_{\max}\mathbf{1}$, and by integrating both sides with respect to η, and using Assumption 1 we have

$$||v^* - v^{\pi_K}||_{1,\eta} \le \gamma^K\frac{2}{1-\gamma}R_{\max} + \sum_{k=0}^{K-1}\gamma^{K-k-1}\sum_{t=0}^{\infty}\gamma^t C_{\eta,\mu}(K-k-1,t)||\epsilon'_{\pi_k}(\pi_{k+1})||_{1,\mu}.$$

The claim follows from the definition of $C_{\eta,\mu}$. □

A.2 Proof of Theorem 3.2

Since the proof of Theorem 3.2 does not depend on the specific critic employed in DPI-Critic, we report its proof in the general form where the e terms depend on the specific Lemma (Lemma 3.2 for LSTD or Lemma 3.3 for BRM) used in the proof.

Proof. The proof follows the same steps as in Theorem 1 in Lazaric et al. (2010a). We prove the following series of inequalities:

$$\begin{aligned}
&\mathcal{L}_k^\Pi(\mu;\pi_{k+1}) \overset{(a)}{\leq} \mathcal{L}_k^\Pi(\widehat{\mu};\pi_{k+1}) + e_0' && \text{w.p. } 1-\delta' \\
&= \frac{1}{N'}\sum_{i=1}^{N'}\Big[Q^{\pi_k}(s^{(i)},a^*) - Q^{\pi_k}\Big(s^{(i)},\pi_{k+1}(s^{(i)})\Big)\Big] + e_0' \\
&\overset{(b)}{\leq} \frac{1}{N'}\sum_{i=1}^{N'}\Big[Q^{\pi_k}(s^{(i)},a^*) - \widehat{Q}^{\pi_k}\Big(s^{(i)},\pi_{k+1}(s^{(i)})\Big)\Big] + e_0' + e_1' + e_2' + e_3 + e_4 && \text{w.p. } 1-2\delta' \\
&\overset{(c)}{\leq} \frac{1}{N'}\sum_{i=1}^{N'}\Big[Q^{\pi_k}(s^{(i)},a^*) - \widehat{Q}^{\pi_k}\Big(s^{(i)},\pi^*(s^{(i)})\Big)\Big] + e_0' + e_1' + e_2' + e_3 + e_4 \\
&\overset{(d)}{\leq} \frac{1}{N'}\sum_{i=1}^{N'}\Big[Q^{\pi_k}(s^{(i)},a^*) - Q^{\pi_k}\Big(s^{(i)},\pi^*(s^{(i)})\Big)\Big] + e_0' + 2(e_1' + e_2' + e_3 + e_4) && \text{w.p. } 1-3\delta' \\
&= \mathcal{L}_k^\Pi(\widehat{\mu};\pi^*) + e_0' + 2(e_1' + e_2' + e_3 + e_4) \\
&\overset{(e)}{\leq} \mathcal{L}_k^\Pi(\mu;\pi^*) + 2(e_0' + e_1' + e_2' + e_3 + e_4) && \text{w.p. } 1-4\delta'
\end{aligned}$$

The statement of the theorem follows by $\delta' = \delta/4$.
(a) It is an immediate application of Lemma 1 in Lazaric et al. (2010a).
(b) This is the result of Lemma 3.2 (Lemma 3.3 for BRM).
(c) From the definition of π_{k+1} in the DPI-Critic algorithm we have

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

thus, $-\frac{1}{N'}\sum_{i=1}^{N'}\widehat{Q}^{\pi_k}\Big(s^{(i)},\pi(s^{(i)})\Big)$ can be maximized by replacing π_{k+1} with any other policy particularly with $\pi^* \in \operatorname{argmin}_{\pi\in\Pi}\mathcal{L}_k^\Pi(\mu;\pi)$.
(d)-(e) These steps follow from Lemma 3.2 and Lemma 1 in Lazaric et al. (2010a). □

B DPI-Critic with Bellman Residual Minimization

In this section we bound the performance of each iteration of DPI-Critic when Bellman Residual Minimization (BRM) is used to train the critic.

Assumption 5. *At each iteration k of DPI-Critic, the critic uses a linear function space spanned by d bounded basis functions (see Section 1.3.1 in Chapter 1). A data-set $\mathcal{D}_k = \Big\{(S_i,R_i,Y_i,Y_i')\Big\}_{i=1}^n$ is built, where $S_i \sim \tau$, $R_i = r\Big(S_i,\pi_k(S_i)\Big)$, and Y_i and Y_i' are two independent states drawn from $P_{\pi_k}(\cdot|S_i)$. Note that here in BRM (unlike LSTD) the sampling distribution τ can be any distribution over the state space.*

Assumption 6. *The rollout set sampling distribution μ is such that for any policy $\pi \in \Pi$ and any action $a \in \mathcal{A}$, $\rho = \mu P_a (P_\pi)^{m-1} \leq C\tau$, where $C < \infty$ is a constant. The distribution ρ is a distribution induced by starting at a state sampled from μ, taking action a, and then following policy π for $m-1$ steps.*

We first report the performance bound for BRM.

Proposition 3.2. (Theorem 7 in Maillard et al. 2010) *Let N samples be collected as in Assumption 5 and $\widehat{v}^\pi$ be the approximation returned by BRM using the linear function space $\mathcal{F}$ as defined in Section 1.3.1 in Chapter 1. Then for any $\delta > 0$, we have*

$$||v^\pi - \widehat{v}^\pi||_{2,\tau} \leq e_{BRM} = \tag{3.9}$$
$$||(I-\gamma P_\pi)^{-1}||_\tau \Bigg((I + \gamma ||P_\pi||_\tau \inf_{f\in\mathcal{F}} ||v^\pi - f||_{2,\tau} + c \Big(\frac{2d \log(2) + 6\log(64|\mathcal{A}|/\delta)}{n} \Big)^{1/4} \Bigg)$$

with probability $1-\delta$, where **(1)** $c = 12[\frac{2}{\xi}(1+\gamma)^2 L^2 + 1] R_{\max}$,
(2) $\xi = \frac{\omega}{||(I-\gamma P_\pi)^{-1}||^2}$,
(3) *$\omega > 0$ is the smallest strictly positive eigenvalue of the Gram matrix with respect to the distribution τ.*

In the following lemma, we bound the difference between the actual action-value function and the one estimated by DPI-Critic.

Lemma 3.3. *Let Assumptions 5 and 6 hold and $\{s^{(i)}\}_{i=1}^{N'}$ be the rollout set with $s^{(i)} \overset{iid}{\sim} \mu$. Let Q^π be the true action-value function of policy π and $\widehat{Q}^\pi$ be its estimation computed by DPI-Critic using M rollouts with rollout size m. Then for any $\delta > 0$, we have*

$$\max_{a\in\mathcal{A}} \left| \frac{1}{N'} \sum_{i=1}^{N'} \left[Q^\pi(s^{(i)}, a) - \widehat{Q}^\pi(s^{(i)}, a) \right] \right| \leq e'_1 + e'_2 + e_3 + e_4,$$

with probability $1-\delta$ (with respect to the random rollout estimates and the random samples in the critic's training set $\mathcal{D}_k$), where

$$e'_1 = (1-\gamma^m) Q_{\max} \sqrt{\frac{2\log(4|\mathcal{A}|/\delta)}{MN'}}, \quad e'_2 = \gamma^m Q_{\max} \sqrt{\frac{2\log(4|\mathcal{A}|/\delta)}{MN'}},$$
$$e_3 = 12\gamma^m B \sqrt{\frac{2\Lambda(N', d, \frac{\delta}{4|\mathcal{A}|M})}{N'}}, \quad e_4 = 2\gamma^m \sqrt{C}\, e_{BRM},$$

with

$$\Lambda(N', d, \delta) = 2(d+1)\log(N') + \log\frac{e}{\delta} + \log\left(9(12e)^{2(d+1)}\right),$$
$$B = Q_{\max}\Big(1 + \frac{2(1-\gamma^2)L^2||(I-\gamma P_{\pi_k})^{-1}||_\tau^2}{\omega}\Big).$$

Proof. We prove the following series of inequalities:

$$
\begin{aligned}
&\Big|\frac{1}{N'}\sum_{i=1}^{N'}[Q^\pi(s^{(i)},a)-\widehat{Q}^\pi(s^{(i)},a)]\Big| \\
&\overset{(a)}{=} \Big|\frac{1}{MN'}\sum_{i=1}^{N'}\sum_{j=1}^{M}[Q^\pi(s^{(i)},a)-R_j^\pi(s^{(i)},a)]\Big| \\
&\overset{(b)}{\le} \Big|\frac{1}{MN'}\sum_{i=1}^{N'}\sum_{j=1}^{M}[Q_m^\pi(s^{(i)},a)-R_j^{\pi,m}(s^{(i)},a)]\Big| + \Big|\frac{\gamma^m}{MN'}\sum_{i=1}^{N'}\sum_{j=1}^{M}\Big[\widehat{v}^\pi(s_m^{(i,j)})- \ \mathbb{E}_{s\sim\nu_i}[v^\pi(s)]\Big]\Big| \\
&\overset{(c)}{\le} e_1' + \Big|\frac{\gamma^m}{MN'}\sum_{i=1}^{N'}\sum_{j=1}^{M}\Big[\widehat{v}^\pi(s_m^{(i,j)})-v^\pi(s_m^{(i,j)})\Big]\Big| + \Big|\frac{\gamma^m}{MN'}\sum_{i=1}^{N'}\sum_{j=1}^{M}\Big[v^\pi(s_m^{(i,j)})- \ \mathbb{E}_{s\sim\nu_i}[v^\pi(s)]\Big]\Big| \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{w.p. } 1-\delta' \\
&\overset{(d)}{\le} e_1'+e_2'+\frac{\gamma^m}{M}\sum_{j=1}^{M}||v^\pi-\widehat{v}^\pi||_{1,\widehat{\rho}_j} \qquad \text{w.p. } 1-2\delta' \\
&\overset{(e)}{\le} e_1'+e_2'+\frac{\gamma^m}{M}\sum_{j=1}^{M}||v^\pi-\widehat{v}^\pi||_{2,\widehat{\rho}_j} \qquad \text{w.p. } 1-2\delta' \\
&\overset{(f)}{\le} e_1'+e_2'+e_3+2\gamma^m||v^\pi-\widehat{v}^\pi||_{2,\rho} \qquad \text{w.p. } 1-3\delta' \\
&\overset{(g)}{\le} e_1'+e_2'+e_3+2\gamma^m\sqrt{C}\ ||v^\pi-\widehat{v}^\pi||_{2,\tau} \\
&\overset{(h)}{\le} e_1'+e_2'+e_3+2\gamma^m\sqrt{C}\ e_{BRM} \qquad \text{w.p. } 1-4\delta'
\end{aligned}
$$

The statement of the lemma is obtained by setting $\delta' = \delta/4$, by taking the union bound over actions.

(a) We use Equation 3.3 to replace $\widehat{Q}^\pi(s^{(i)},a)$.
(b) We replace $R_j^\pi(s^{(i)},a)$ from Equation 3.1 and use the fact that $Q^\pi(s^{(i)},a) = Q_m^\pi(s^{(i)},a)+\gamma^m\mathbb{E}_{s\sim\nu_i}[v^\pi(s)]$, where $Q_m^\pi(s^{(i)},a)=\mathbb{E}\Big[r(s^{(i)},a)+\sum_{t=1}^{m-1}\gamma^t r\Big(s_t^{(i)},\pi(s_t^{(i)})\Big)\Big]$ and $\nu_i = \mathbb{I}(s^{(i)})P_a(P_\pi)^{m-1}$ is the distribution over states induced by starting at state $s^{(i)}$, taking action a, and then following the policy π for $m-1$ steps. We split the sum using the triangle inequality.
(c) Using the Chernoff-Hoeffding inequality, with probability $1-\delta'$ (with respect to the random samples used to build the rollout estimates), we have

$$\Big|\frac{1}{MN'}\sum_{i=1}^{N'}\sum_{j=1}^{M}[Q_m^\pi(s^{(i)},a)-R_j^{\pi,m}(s^{(i)},a)]\Big| \le e_1' = (1-\gamma^m)Q_{\max}\sqrt{\frac{2\log(1/\delta')}{MN'}}.$$

(d) Using the Chernoff-Hoeffding inequality, with probability $1-\delta'$ (with respect to the last state achieved by the rollouts trajectories), we have

$$\Big|\frac{\gamma^m}{MN'}\sum_{i=1}^{N'}\sum_{j=1}^{M}\Big[v^\pi(s_m^{(i,j)})-\mathbb{E}_{s\sim\nu_i}[v^\pi(s)]\Big]\Big| \le e_2' = \gamma^m Q_{\max}\sqrt{\frac{2\log(1/\delta')}{MN'}}.$$

We also use the definition of empirical ℓ_1-norm and replace the second term with $||v^\pi - \widehat{v}^\pi||_{1,\widehat{\rho}_j}$, where $\widehat{\rho}_j$ is the empirical distribution corresponding to the distribution $\rho = \mu P_a (P_\pi)^{m-1}$. In fact for any $1 \leq j \leq M$, the samples $s_m^{(i,j)}$ are i.i.d. from ρ.
(e) We move from ℓ_1-norm to ℓ_2-norm: for any $s \in \mathbb{R}^n$, using the Cauchy-Schwarz inequality, we obtain

$$||s||_{1,\widehat{\rho}} = \frac{1}{N}\sum_{i=1}^{N}|s^{(i)}| \leq \sqrt{\sum_{i=1}^{N}\frac{1}{N}}\sqrt{\sum_{i=1}^{N}\frac{1}{N}|s^{(i)}|^2} = ||s||_{2,\widehat{\rho}}.$$

(f) We note that $\widehat{v}$ is a random variable independent from the samples used to build the rollout estimates. Thus, applying Corollary 12 in Lazaric et al. (2012), for any j we have

$$||v^\pi - \widehat{v}^\pi||_{2,\widehat{\rho}_j} \leq 2||v^\pi - \widehat{v}^\pi||_{2,\rho} + e_3(\delta'')$$

with probability $1-\delta''$ (with respect to the samples in $\widehat{\rho}_j$) and $e_3(\delta'') = 12B\sqrt{\frac{2\Lambda(N',d,\delta'')}{N'}}$. Using the upper bound on the solutions returned by BRM, when the number of samples N is large enough, then with high probability (see Corollary 5 in Maillard et al. 2010 for details)

$$B = Q_{\max}\Big(1 + \frac{2(1-\gamma^2)L^2||(I-\gamma P_{\pi_k})^{-1}||_\tau^2}{\omega}\Big).$$

Finally, by taking a union bound over all j's and setting $\delta'' = \delta'/M$, we obtain the definition of e_3 in the final statement. **(g)** Using Assumption 6, we have $||v^\pi - \widehat{v}||_{2,\rho} \leq \sqrt{C}||v^\pi - \widehat{v}||_{2,\tau}$.
(h) Here, we simply replace $||v^\pi - \widehat{v}||_{2,\tau}$ with the bound in Proposition 3.2. □

Chapter 4
Approximate Modified Policy Iteration

In this chapter,[1] we study the approximate version of the modified policy iteration (MPI) algorithm (Puterman and Shin, 1978). As discussed in Section 1.2 of Chapter 1, MPI is a dynamic programming (DP) algorithm that contains the two celebrated policy and value iteration methods. Despite its generality, MPI has not been thoroughly studied, especially its approximation form which is used when the state and/or action spaces are large or infinite. In this chapter, we propose three implementations of approximate MPI (AMPI) that are extensions of the well-known approximate DP algorithms. The first two correspond to fitted-value iteration and fitted-Q iteration, and the last one is a classification-based DP algorithm. We first provide error propagation analysis for AMPI that unifies those for approximate policy and value iteration algorithms. We then develop the finite-sample performance analysis of our proposed AMPI algorithms, which highlights the influence of their free parameters. Finally, we illustrate and evaluate the behavior of the new algorithms in the Mountain Car and Tetris problems.

Contents

[1]This chapter is an extended version of our ICML (Scherrer et al., 2012) and NIPS (Gabillon et al., 2013) papers, which has also been submitted to JMLR (Scherrer et al., 2014).

1 Introduction

The aim of this chapter is to show that, similar to its exact form, approximate MPI (AMPI), discussed in Section 1.2 of Chapter 1, may represent an interesting alternative to AVI and API algorithms. In problems with large state and/or action spaces, approximate versions of VI (AVI) and PI (API) have been the focus of a rich literature (see e.g., Bertsekas and Tsitsiklis 1996, Szepesvári 2010). Approximate VI (AVI) generates the next value function as the approximation of the application of the Bellman optimality operator to the current value (Ernst et al., 2005, Antos et al., 2007, Munos and Szepesvári, 2008). On the other hand, approximate PI (API) first finds an approximation of the value of the current policy and then generates the next policy as greedy w.r.t. this approximation (Bertsekas and Tsitsiklis, 1996, Munos, 2003, Lagoudakis and Parr, 2003a). Another related algorithm is λ-policy iteration (Bertsekas and Ioffe, 1996), which is a rather complicated variation of MPI. It involves computing a fixed-point at each iteration, and thus, suffers from some of the drawbacks of this class of PI algorithms. The approximate λ-policy iteration algorithm has been analyzed by Thiéry and Scherrer (2010a) (see also Scherrer 2013).

In this chapter, we propose three implementations of AMPI (Section 2) that generalize the AVI implementations of Ernst et al. (2005), Antos et al. (2007), Munos and Szepesvári (2008) and the classification-based API algorithms of Lagoudakis and Parr (2003b), Fern et al. (2006), Lazaric et al. (2010a) as well as the DPI-Critic algorithm (Gabillon et al., 2011b) proposed and analyzed in Chapter 3. We then provide an error propagation analysis for AMPI (Section 3), which shows how the ℓ_p-norm of its performance loss can be controlled by the error at each iteration of the algorithm. We show that the error propagation analysis of AMPI is more involved than that of AVI and API. This is due to the fact that neither the contraction nor monotonicity arguments, that the error propagation analysis of these two algorithms rely on, hold for AMPI. The analysis of this section unifies those for AVI and API and is applied to the AMPI implementations presented in Section 2. In Section 4, we provide finite sample performance analysis for the three algorithms of Section 2. The analysis of the classification-based implementation of MPI (CBMPI) reveals results similar to those for DPI-Critic, which indicate that the parameter m allows us to balance the estimation error of the classifier with the overall quality of the value function approximation, and plays a similar role as the length of the rollouts in DPI-Critic. Finally, we evaluate the proposed algorithms and compare them with several existing methods in the classic Mountain Car problem as well as in the challenging game of Tetris in Section 5. In fact, the game of Tetris is a domain were all the results reported for ADP methods are worse by several orders of magnitude than the state of the art approach, which is based on the cross entropy method (Rubinstein and Kroese, 2004). Those ADP methods were uniquely based on approximating the value function while the cross entropy method directly searches in the policy space. Therefore, we conjecture that an ADP that searches in the policy space could finally perform well in the game of Tetris. To test this conjecture we used CBMPI, an algorithm that performs an approximate greedy step with the use of a classifier that searches in fixed policy space.

2 Approximate MPI Algorithms

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

2.1 AMPI-V

The first and simplest AMPI algorithm presented in this chapter called AMPI-V. Figure 4.1 contains the pseudocode of this algorithm. In AMPI-V, we assume that the values v_k are represented in a function space $\mathcal{F} \subseteq \mathbb{R}^{|\mathcal{S}|}$. In any state s, the action $\pi_{k+1}(s)$ that is greedy w.r.t. v_k can be estimated as follows:

$$\pi_{k+1}(s) = \arg\max_{a \in \mathcal{A}} \frac{1}{M} \sum_{j=1}^{M} \Big(r_a^{(j)} + \gamma v_k(s_a^{(j)}) \Big), \tag{4.1}$$

where $\forall a \in \mathcal{A}$ and $1 \le j \le M$, $r_a^{(j)}$ and $s_a^{(j)}$ are samples of rewards and next states when action a is taken in state s. Thus, approximating the greedy action in a state s requires $M|\mathcal{A}|$ samples. The algorithm works as follows. We samples N states from a distribution μ on $\mathcal{S}$, and build a rollout set $\mathcal{D}_k = \{s^{(i)}\}_{i=1}^N$, $s^{(i)} \sim \mu$. From each state $s^{(i)} \in \mathcal{D}_k$, we generate a rollout of size m, i.e., $\Big(s^{(i)}, a_0^{(i)}, r_0^{(i)}, s_1^{(i)}, \dots, a_{m-1}^{(i)}, r_{m-1}^{(i)}, s_m^{(i)}\Big)$, where $a_t^{(i)}$ is the action suggested by π_{k+1} in state $s_t^{(i)}$, computed using Equation 4.1, and $r_t^{(i)}$ and $s_{t+1}^{(i)}$ are the reward and next state induced by this choice of action. For each $s^{(i)}$, we then compute a rollout estimate

$$\widehat{v}_{k+1}(s^{(i)}) = \sum_{t=0}^{m-1} \gamma^t r_t^{(i)} + \gamma^m v_k(s_m^{(i)}), \tag{4.2}$$

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

$$\widehat{\mathcal{L}}_k^{\mathcal{F}}(\widehat{\mu}; v) = \frac{1}{N} \sum_{i=1}^{N} \Big(\widehat{v}_{k+1}(s^{(i)}) - v(s^{(i)}) \Big)^2, \tag{4.3}$$

with the goal of minimizing the true error

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

Each iteration of AMPI-V requires N rollouts of size m, and in each rollout, each of the $|\mathcal{A}|$ actions needs M samples to compute Equation 4.1. This gives a total of $Nm(M|\mathcal{A}| + 1)$ transition samples. Note that the fitted value iteration algorithm (Munos and Szepesvári, 2008) is a special case of AMPI-V when $m = 1$.

```
Input: Value function space $\mathcal{F}$, state distribution $\mu$
Initialize: Let $v_0 \in \mathcal{F}$ be an arbitrary value function
for $k = 0, 1, \dots$ do
  • Perform rollouts:
  Construct the rollout set $\mathcal{D}_k = \{s^{(i)}\}_{i=1}^N$, $s^{(i)} \overset{\text{iid}}{\sim} \mu$
  for all states $s^{(i)} \in \mathcal{D}_k$ do
    Perform a rollout (using Equation 4.1 for each action)
    $\widehat{v}_{k+1}(s^{(i)}) = \sum_{t=0}^{m-1} \gamma^t r_t^{(i)} + \gamma^m v_k(s_m^{(i)})$
  end for
  • Approximate value function:
  $v_{k+1} \in \underset{v \in \mathcal{F}}{\operatorname{argmin}} \widehat{\mathcal{L}}_k^{\mathcal{F}}(\widehat{\mu}; v)$    (regression)    (see Equation 4.3)
end for
```

Figure 4.1: The pseudo-code of the AMPI-V algorithm.

2.2 AMPI-Q

In AMPI-Q, we replace the value function $v : \mathcal{S} \to \mathbb{R}$ with the action-value function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Figure 4.2 contains the pseudocode of this algorithm. The Bellman operator for a policy π at a state-action pair (s, a) can then be written as

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

and the greedy operator is defined as

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

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

$$\pi_{k+1}(s) \in \arg\max_{a \in \mathcal{A}} \; Q_k(s,a). \tag{4.4}$$

The *evaluation step* is similar to that of AMPI-V, with the difference that now we work with state-action pairs. We sample N state-action pairs from a distribution μ on $\mathcal{S} \times \mathcal{A}$ and build a rollout set $\mathcal{D}_k = \{(s^{(i)}, a^{(i)})\}_{i=1}^N$, $(s^{(i)}, a^{(i)}) \sim \mu$. For each $(s^{(i)}, a^{(i)}) \in \mathcal{D}_k$, we generate a rollout of size m, i.e., $\Big(s^{(i)}, a^{(i)}, r_0^{(i)}, s_1^{(i)}, a_1^{(i)}, \cdots, s_m^{(i)}, a_m^{(i)}\Big)$, where the first action is $a^{(i)}$, $a_t^{(i)}$ for $t \geq 1$ is the action suggested by π_{k+1} in state $s_t^{(i)}$ computed using Equation 4.4, and $r_t^{(i)}$ and $s_{t+1}^{(i)}$ are the reward and next state induced by this choice of action. For each $(s^{(i)}, a^{(i)}) \in \mathcal{D}_k$, we then compute the rollout estimate

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

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

$$\widehat{\mathcal{L}}_k^{\mathcal{F}}(\widehat{\mu}; Q) = \frac{1}{N} \sum_{i=1}^N \Big(\widehat{Q}_{k+1}(s^{(i)}, a^{(i)}) - Q(s^{(i)}, a^{(i)})\Big)^2, \tag{4.5}$$

```
Input: Value function space $\mathcal{F}$, state distribution $\mu$
Initialize: Let $Q_0 \in \mathcal{F}$ be an arbitrary value function
for $k = 0, 1, \dots$ do
  • Perform rollouts:
  Construct the rollout set $\mathcal{D}_k = \{(s^{(i)}, a^{(i)}\}_{i=1}^N$, $(s^{(i)}, a^{(i)}) \overset{\text{iid}}{\sim} \mu$
  for all states $(s^{(i)}, a^{(i)}) \in \mathcal{D}_k$ do
    Perform a rollout (using Equation 4.4 for each action)
    $\widehat{Q}_{k+1}(s^{(i)}, a^{(i)}) = \sum_{t=0}^{m-1} \gamma^t r_t^{(i)} + \gamma^m Q_k(s_m^{(i)}, a_m^{(i)})$,
  end for
  • Approximate action-value function:
  $Q_{k+1} \in \underset{Q\in\mathcal{F}}{\text{argmin}}\, \widehat{\mathcal{L}}_k^{\mathcal{F}}(\widehat{\mu}; Q)$      (regression)      (see Equation 4.5)
end for
```

Figure 4.2: The pseudo-code of the AMPI-Q algorithm.

with the goal of minimizing the true error

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

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

2.3 Classification-based MPI

The third AMPI algorithm presented in this chapter, called classification-based MPI (CBMPI), uses an explicit representation for the policies π_k, in addition to the one used for the value functions v_k. The idea is similar to the classification-based PI algorithms (Lagoudakis and Parr, 2003b, Fern et al., 2006, Lazaric et al., 2010a) discussed in Section 1.3.2 of Chapter 2 and also the DPI-Critic algorithm, introduced in Chapter 3. In the classification-based policy iteration algorithms, at each iteration, we search for the greedy policy in a policy space Π (defined by a classifier) instead of computing it from the estimated value or action-value function (like in AMPI-V and AMPI-Q).
In order to describe CBMPI, we first rewrite the MPI formulation (Equations 2.7 and 2.8) as

$$v_k = (T_{\pi_k})^m v_{k-1} \qquad \text{(evaluation step)} \qquad (4.6)$$

$$\pi_{k+1} = \mathcal{G}\left[(T_{\pi_k})^m v_{k-1}\right] \qquad \text{(greedy step)} \qquad (4.7)$$

Note that in the new formulation both v_k and π_{k+1} are functions of $(T_{\pi_k})^m v_{k-1}$. CBMPI is an approximate version of this new formulation. As described in Figure 4.3, CBMPI

```
Input: Value function space $\mathcal{F}$, policy space $\Pi$, state distribution $\mu$
Initialize: Let $\pi_1 \in \Pi$ be an arbitrary policy and $v_0 \in \mathcal{F}$ an arbitrary value function
for $k = 1, 2, \dots$ do
  • Perform rollouts:
  Construct the rollout set $\mathcal{D}_k = \{s^{(i)}\}_{i=1}^N$, $s^{(i)} \overset{\text{iid}}{\sim} \mu$
  for all states $s^{(i)} \in \mathcal{D}_k$ do
    Perform a rollout and return $\widehat{v}_k(s^{(i)})$          (using Equation 4.8)
  end for
  Construct the rollout set $\mathcal{D}'_k = \{s^{(i)}\}_{i=1}^{N'}$, $s^{(i)} \overset{\text{iid}}{\sim} \mu$
  for all states $s^{(i)} \in \mathcal{D}'_k$ and actions $a \in \mathcal{A}$ do
    for $j = 1$ to $M$ do
      Perform a rollout and return $R_k^j(s^{(i)}, a)$          (using Equation 4.13)
    end for
    $\widehat{Q}_k(s^{(i)}, a) = \frac{1}{M}\sum_{j=1}^M R_k^j(s^{(i)}, a)$
  end for
  • Approximate value function:
  $v_k \in \underset{v \in \mathcal{F}}{\operatorname{argmin}}\, \widehat{\mathcal{L}}_k^{\mathcal{F}}(\widehat{\mu}; v)$          (regression)     (see Equation 4.9)
  • Approximate greedy policy:
  $\pi_{k+1} \in \underset{\pi \in \Pi}{\operatorname{argmin}}\, \widehat{\mathcal{L}}_k^{\Pi}(\widehat{\mu}; \pi)$          (classification)     (see Equation 4.14)
end for
```

Figure 4.3: The pseudo-code of the CBMPI algorithm.

begins with arbitrary initial policy $\pi_1 \in \Pi$ and value function $v_0 \in \mathcal{F}$.[2] At each iteration k, a new value function v_k is built as the best approximation of the m-step Bellman operator $(T_{\pi_k})^m v_{k-1}$ in $\mathcal{F}$ (*evaluation step*). This is done by solving a regression problem whose target function is $(T_{\pi_k})^m v_{k-1}$. To set up the regression problem, we build a rollout set $\mathcal{D}_k$ by sampling N states i.i.d. from a distribution μ.[3] For each state $s^{(i)} \in \mathcal{D}_k$, we generate a rollout $\left(s^{(i)}, a_0^{(i)}, r_0^{(i)}, s_1^{(i)}, \dots, a_{m-1}^{(i)}, r_{m-1}^{(i)}, s_m^{(i)}\right)$ of size m, where $a_t^{(i)} = \pi_k(s_t^{(i)})$, and $r_t^{(i)}$ and $s_{t+1}^{(i)}$ are the reward and next state induced by this choice of action. From this rollout, we compute an unbiased estimate $\widehat{v}_k(s^{(i)})$ of $\left[(T_{\pi_k})^m v_{k-1}\right](s^{(i)})$ as in Equation 4.2:

$$\widehat{v}_k(s^{(i)}) = \sum_{t=0}^{m-1} \gamma^t r_t^{(i)} + \gamma^m v_{k-1}(s_m^{(i)}), \tag{4.8}$$

and use it to build a training set $\left\{\left(s^{(i)}, \widehat{v}_k(s^{(i)})\right)\right\}_{i=1}^N$. This training set is then used by the regressor to compute v_k as an estimate of $(T_{\pi_k})^m v_{k-1}$. Similar to the AMPI-V

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

[3]Here we used the same sampling distribution μ for both regressor and classifier, but in general different distributions may be used for these two components of the algorithm.

algorithm, the regressor here finds a function $v \in \mathcal{F}$ that minimizes the empirical error

$$\widehat{\mathcal{L}}_k^{\mathcal{F}}(\widehat{\mu}; v) = \frac{1}{N}\sum_{i=1}^{N}\Big(\widehat{v}_k(s^{(i)}) - v(s^{(i)})\Big)^2, \tag{4.9}$$

with the goal of minimizing the true error

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

In a very similar manner as in DPI-Critic (see Equation 3.4), the *greedy step* at iteration k computes the policy π_{k+1} as the best approximation of $\mathcal{G}\Big[(T_{\pi_k})^m v_{k-1}\Big]$ by solving a cost-sensitive classification problem. From the definition of a greedy policy, if $\pi = \mathcal{G}\Big[(T_{\pi_k})^m v_{k-1}\Big]$, for each $s \in \mathcal{S}$, we have

$$\Big[T_\pi (T_{\pi_k})^m v_{k-1}\Big](s) = \max_{a\in\mathcal{A}} \Big[T_a (T_{\pi_k})^m v_{k-1}\Big](s). \tag{4.10}$$

By defining $Q_k(s,a) = \Big[T_a (T_{\pi_k})^m v_{k-1}\Big](s)$, we may rewrite Equation 4.10 as

$$Q_k\Big(s, \pi(s)\Big) = \max_{a\in\mathcal{A}} Q_k(s,a). \tag{4.11}$$

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

$$\mathcal{L}_{\pi_k, v_{k-1}}^{\Pi}(\mu; \pi) = \int \Big[\max_{a\in\mathcal{A}} Q_k(s,a) - Q_k\Big(s, \pi(s)\Big)\Big]\mu(ds). \tag{4.12}$$

To simplify notation we use $\mathcal{L}_k^{\Pi}$ instead of $\mathcal{L}_{\pi_k, v_{k-1}}^{\Pi}$. Note that here we deliberately use the same notations as in the DPI-Critic chapter in order to highlight the similarities between DPI-Critic and CBMPI. To set up this cost-sensitive classification problem, we build a rollout set $\mathcal{D}'_k$ by sampling N' states i.i.d. from a distribution μ. For each state $s^{(i)} \in \mathcal{D}'_k$ and each action $a \in \mathcal{A}$, we build M independent rollouts of size $m+1$, i.e.,[4]

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

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

$$R_k^j(s^{(i)}, a) = \sum_{t=0}^{m} \gamma^t r_t^{(i,j)} + \gamma^{m+1} v_{k-1}(s_{m+1}^{(i,j)}). \tag{4.13}$$

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

$$\widehat{\mathcal{L}}_k^{\Pi}(\widehat{\mu}; \pi) = \frac{1}{N'}\sum_{i=1}^{N'}\Big[\max_{a\in\mathcal{A}} \widehat{Q}_k(s^{(i)}, a) - \widehat{Q}_k\Big(s^{(i)}, \pi(s^{(i)})\Big)\Big], \tag{4.14}$$

[4]We may implement CBMPI more sample efficient by reusing the rollouts generated for the greedy step in the evaluation step, but this makes the analysis of the algorithm more complicated.

with the goal of minimizing the true error $\mathcal{L}_k^{\Pi}(\mu;\pi)$ defined by Equation 4.12.

Each iteration of CBMPI requires $Nm + M|\mathcal{A}|N'(m+1)$ (or $M|\mathcal{A}|N'(m+1)$ in case we reuse the rollouts, see Footnote 3) transition samples. Note that when m tends to ∞, we recover the DPI algorithm proposed and analyzed by Lazaric et al. (2010a).

Remark 4.1 (Comparison between CBMPI and DPI-Critic). The DPI-Critic algorithm, described in Chapter 3, and CBMPI are similar algorithms. They are both CBPI algorithms with the use of a critic which compensates for the bias coming from the truncation of the rollouts. Moreover, as they both compute the successive greedy policies by minimizing similar losses (see Equations 3.4 and 4.14), their corresponding theoretical guarantees have the same shape. Here, we highlight some of their differences. **1)** They do not belong to the same class of algorithms: DPI-Critic is typically a PI algorithm while CBMPI belongs to the wider class of MPI algorithms, and thus, their analyses are different. **2)** Their different ways of building the critic may result in different behavior in practice. Upon convergence, both critics are expected to converge to v^*. However, when building the critic, the target function of DPI-Critic is v^{π_k} while the target function of CBMPI is $(T_{\pi_k})^m v_{k-1}$. Therefore, we have noticed in our experiments that the value function approximation built in CBMPI, which follows a VI procedure, takes more iterations to converge to a good approximation of v^* than the value function built in our implementation of DPI-Critic, which solves a fixed-point problem, namely using LSTD. This phenomenon highly depends on the parameter m, the shape of v^*, and on the choice of v_0. Another difference between the two algorithms is that, in DPI-Critic, the critic is built exclusively on fresh new samples at each iteration while in CBMPI, the iterative procedure of VI builds the new critic based on the new samples plus the previous critic. In goal-based problems such as in Mountain Car (MC) and Inverted Pendulum (IP) (see Section 5 of Chapter 3), where the associated reward function is only informative at terminal states, it is critical to observe transitions to the terminal states in order to build a non trivial value function approximation. If those are not observed in one iteration, DPI-Critic is more prone to compute poor rollout estimates and fail to compute the greedy policy while CBMPI, which reuses its previous critic, is likely to be more stable in that respect.

2.4 Possible Approaches to Reuse the Samples

In all the proposed AMPI algorithms, we generate fresh samples for the rollouts, and even for the starting states, at each iteration. This results in high sample complexity for these algorithms. In this section, we propose two possible approaches to circumvent this problem and to keep the number of samples independent of the number of iterations. One approach would be to use a fixed set of starting samples $(s^{(i)})$ or $(s^{(i)}, a^{(i)})$ for all iterations, and think of a tree of depth m that contains all the possible outcomes of m-steps choices of actions (this tree contains $|\mathcal{A}|^m$ leaves). This is in a way reminiscent of the work by Kearns et al. (2000). Using this tree, all the trajectories with the same actions share the same samples. In practice, it is not necessarily to build the entire depth m tree, it is only needed to add a branch when the desired action does not belong to the tree. Using this approach, the sample complexity of the algorithm no longer depends on the number of iterations. For example, we may only need $NM|\mathcal{A}|^m$ transitions for the CBMPI algorithm. We may also consider the case where we do not have access to a generative model of the system, and all we have is a set of trajectories of size m generated by a behavior policy π_b that is assumed to choose all actions a in each state s with a positive probability (i.e., $\pi_b(a|s) > 0,\ \forall s, \forall a$) (Precup et al., 2000, 2001). In this case, one may still compute an unbiased estimate of the application of $(T_\pi)^m$ operator to value and action-value functions. For instance, given a m-step sample trajectory $(s, a_0, r_0, s_1, \dots, s_m, a_m)$ generated by π_b, an unbiased estimate of $[(T_\pi)^m v](s)$ may be computed as (assuming that the distribution μ has the following factored form $p(s, a_0|\mu) = p(s)\pi_b(a_0|s)$ at state s)

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

is an importance sampling correction factor that can be computed along the trajectory. However, this process may significantly increase the variance of such an estimate, and thus, require many more samples.

3 Error Propagation

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

Proposition 4.1. *If $m > 1$, there exists no norm for which the operator that MPI uses to update the values from one iteration to the next is a contraction.*

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

We also know that the analysis of PI usually relies on the fact that the sequence of the generated values is non-decreasing (Bertsekas and Tsitsiklis, 1996, Munos, 2003). Unfortunately, it can be easily shown that for m finite, the value functions generated by MPI may decrease (it suffices to take a very high initial value). It can be seen from what we just described and from Proposition 4.1 that for $m \neq 1$ and ∞, MPI is neither contracting nor non-decreasing, and thus, a new proof is needed for the propagation of error in this algorithm.

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

$$\pi_k = \widehat{\mathcal{G}}_{\epsilon'_k} v_{k-1}. \tag{4.15}$$

Equation 4.15 means that for any policy π', we have $T_{\pi'} v_{k-1} \leq T_{\pi_k} v_{k-1} + \epsilon'_k$. AMPI then generates the new value function v_k with some error ϵ_k, called the *evaluation step error*

$$v_k = (T_{\pi_k})^m v_{k-1} + \epsilon_k. \tag{4.16}$$

Before showing how these two errors are propagated through the iterations of AMPI, let us first define them in the context of each of the algorithms presented in Section 2 separately.

AMPI-V: ϵ_k is the error in fitting the value function v_k. This error can be further decomposed into two parts: the one related to the approximation power of $\mathcal{F}$ and the one due to the finite number of samples/rollouts. ϵ'_k is the error due to using a finite number of samples M for estimating the greedy actions.

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

CBMPI: This algorithm iterates as follows:

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

Unfortunately, this does not exactly match with the model described in Equations 4.15 and 4.16. By introducing the auxiliary variable $w_k \triangleq (T_{\pi_k})^m v_{k-1}$, we have $v_k = w_k + \epsilon_k$, and thus, we may write

$$\pi_{k+1} = \widehat{\mathcal{G}}_{\epsilon'_{k+1}} \left[w_k\right]. \tag{4.17}$$

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

$$w_k = (T_{\pi_k})^m v_{k-1} = (T_{\pi_k})^m (w_{k-1} + \epsilon_{k-1}) = (T_{\pi_k})^m w_{k-1} + (\gamma P_{\pi_k})^m \epsilon_{k-1}. \tag{4.18}$$

Now, Equations 4.17 and 4.18 exactly match Equations 4.15 and 4.16 by replacing v_k with w_k and ϵ_k with $(\gamma P_{\pi_k})^m \epsilon_{k-1}$.

The rest of this section is devoted to show how the errors ϵ_k and ϵ'_k propagate through the iterations of an AMPI algorithm. We only outline the main arguments that

will lead to the performance bound of Theorem 4.1 and report most proofs in Sections A to D. Here we follow the line of analysis developed by Thiéry and Scherrer (2010c). The results are obtained using the following three quantities:
1) The distance between the optimal value function and the value before approximation at the k^{th} iteration: $d_k \triangleq v^* - (T_{\pi_k})^m v_{k-1} = v^* - (v_k - \epsilon_k)$.
2) The shift between the value before approximation and the value of the policy at the k^{th} iteration: $s_k \triangleq (T_{\pi_k})^m v_{k-1} - v^{\pi_k} = (v_k - \epsilon_k) - v^{\pi_k}$.
3) The Bellman residual at the k^{th} iteration: $b_k \triangleq v_k - T_{\pi_{k+1}} v_k$.

We are interested in finding an upper bound on the **loss** $l_k \triangleq v^* - v^{\pi_k} = d_k + s_k$. To do so, we will upper bound d_k and s_k, which requires a bound on the Bellman residual b_k. More precisely, the core of our analysis is to prove the following point-wise inequalities for our three quantities of interest.

Lemma 4.1. *Let $k \geq 1$, $x_k \triangleq (I - \gamma P_{\pi_k})\epsilon_k + \epsilon'_{k+1}$ and $y_k \triangleq -\gamma P_{\pi^*}\epsilon_k + \epsilon'_{k+1}$. We have:*

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

Proof. See Section A. □

Since the stochastic kernels are non-negative, the bounds in Lemma 4.1 indicate that the loss l_k will be bounded if the errors ϵ_k and ϵ'_k are controlled. In fact, if we define ϵ as a uniform upper-bound on the errors $|\epsilon_k|$ and $|\epsilon'_k|$, the first inequality in Lemma 4.1 implies that $b_k \leq O(\epsilon)$, and as a result, the second and third inequalities gives us $d_k \leq O(\epsilon)$ and $s_k \leq O(\epsilon)$. This means that the loss will also satisfy $l_k \leq O(\epsilon)$.

Our bound for the loss l_k is the result of careful expansion and combination of the three inequalities in Lemma 4.1. Before we state this result, we introduce some notations that will ease our formulation.

Definition 4.1. *For a positive integer n, we define $\mathbb{P}_n$ as the set of transition kernels that are defined as follows:*

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

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

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

Using the notation in Definition 4.1, we now derive a point-wise bound on the loss.

Lemma 4.2. *After k iterations, the losses of AMPI-V and AMPI-Q satisfy*

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

while the loss of CBMPI satisfies

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

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

Proof. See Appendix B. □

Remark 4.2. A close look at the existing point-wise error bounds for AVI (Munos, 2007, Lemma 4.1) and API (Munos, 2003, Corollary 10) shows that they do not consider error in the greedy step (i.e., $\epsilon'_k = 0$) and that they have the following form:

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

This indicates that the bound in Lemma 4.2 not only unifies the analysis of AVI and API, but it generalizes them to the case of error in the greedy step and to a finite horizon k. Moreover, our bound suggests that the way the errors are propagated in the whole family of algorithms VI/PI/MPI does not depend on m at the level of the abstraction suggested by Definition 4.1.[5]

The next step is to show how the point-wise bound of Lemma 4.2 can turn to a bound in weighted ℓ_p-norm, which for any function $f : \mathcal{S} \to \mathbb{R}$ and any distribution μ on $\mathcal{S}$ is defined as $\|f\|_{p,\mu} \triangleq \left[\int |f(x)|^p\mu(dx)\right]^{1/p}$. Munos (2003, 2007), Munos and Szepesvári (2008), and the recent work of Farahmand et al. (2010), which provides the most refined bounds for API and AVI, show how to do this process through quantities, called *concentrability coefficients*, that measure how a distribution over states may concentrate through the dynamics of the MDP. We now state a lemma that generalizes the analysis of Farahmand et al. (2010) to a larger class of concentrability coefficients. We will discuss the potential advantage of this new class in Remark 4.5. We will also show through the proofs of Theorems 4.1 and 4.3, how the result of Lemma 4.3 provides us with a flexible tool for turning point-wise bounds into ℓ_p-norm bounds. Theorem 4.3 in Section D provides an alternative bound for the loss of AMPI, which in analogy with the results of Farahmand et al. (2010) shows that the last iterations have the highest impact on the loss (the influence exponentially decreases towards the initial iterations).

Lemma 4.3. *Let $\mathcal{I}$ and $(\mathcal{J}_i)_{i\in\mathcal{I}}$ be sets of positive integers, $\{\mathcal{I}_1,\dots,\mathcal{I}_n\}$ be a partition of $\mathcal{I}$, and f and $(g_i)_{i\in\mathcal{I}}$ be functions satisfying*

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

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

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

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

with the following concentrability coefficients

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

with the Radon-Nikodym derivative based quantity

$$c_q(j) \triangleq \max_{\pi_1,\cdots,\pi_j} \left\| \frac{d(\rho P_{\pi_1} P_{\pi_2}\cdots P_{\pi_j})}{d\mu} \right\|_{q,\mu}. \tag{4.19}$$

Proof. See Appendix C. □

We now derive a ℓ_p-norm bound for the loss of the AMPI algorithm by applying Lemma 4.3 to the point-wise bound of Lemma 4.2.

Theorem 4.1. *Let ρ and μ be distributions over states. Let p, q, and q' be such that $\frac{1}{q}+\frac{1}{q'}=1$. After k iterations, the loss of AMPI satisfies*

$$\|l_k\|_{p,\rho} \leq \frac{2(\gamma-\gamma^k)\Big(\mathcal{C}_q^{1,k,0}\Big)^{\frac{1}{p}}}{(1-\gamma)^2} \sup_{1\leq j\leq k-1} \|\epsilon_j\|_{pq',\mu} + \frac{(1-\gamma^k)\Big(\mathcal{C}_q^{0,k,0}\Big)^{\frac{1}{p}}}{(1-\gamma)^2} \sup_{1\leq j\leq k} \|\epsilon'_j\|_{pq',\mu} + g(k), \tag{4.20}$$

while the loss of CBMPI satisfies

$$\|l_k\|_{p,\rho} \leq \frac{2\gamma^m(\gamma-\gamma^{k-1})\Big(\mathcal{C}_q^{2,k,m}\Big)^{\frac{1}{p}}}{(1-\gamma)^2} \sup_{1\leq j\leq k-2} \|\epsilon_j\|_{pq',\mu} + \frac{(1-\gamma^k)\Big(\mathcal{C}_q^{1,k,0}\Big)^{\frac{1}{p}}}{(1-\gamma)^2} \sup_{1\leq j\leq k} \|\epsilon'_j\|_{pq',\mu} + g(k), \tag{4.21}$$

where for all q, l, k and d, the concentrability coefficients $\mathcal{C}_q^{l,k,d}$ are defined as

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

with $c_q(j)$ given by Equation 4.19, and $g(k)$ is defined as

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

Proof. See Appendix D. □

Remark 4.3. When p tends to infinity, the first bound of Theorem 4.1 reduces to

$$\|l_k\|_\infty \le \frac{2(\gamma-\gamma^k)}{(1-\gamma)^2} \sup_{1\le j\le k-1} \|\epsilon_j\|_\infty + \frac{1-\gamma^k}{(1-\gamma)^2} \sup_{1\le j\le k} \|\epsilon'_j\|_\infty + \frac{2\gamma^k}{1-\gamma} \min(\|d_0\|_\infty, \|b_0\|_\infty). \tag{4.22}$$

When k goes to infinity, Equation 4.22 gives us a generalization of the API ($m=\infty$) bound of Bertsekas and Tsitsiklis (1996, Proposition 6.2), i.e.,

$$\limsup_{k\to\infty} \|l_k\|_\infty \le \frac{2\gamma \sup_{1\le j\le k-1} \|\epsilon_j\|_\infty + \sup_{1\le j\le k} \|\epsilon'_j\|_\infty}{(1-\gamma)^2}.$$

Moreover, since our point-wise analysis generalizes those of API and AVI (as noted in Remark 4.2), the ℓ_p-bound of Equation 4.20 unifies and generalizes those for API (Munos, 2003) and AVI (Munos, 2007).

Remark 4.4. The arguments we developed globally follow those originally developed for λ-policy iteration (Scherrer, 2013). With respect to that work, our proof is significantly simpler thanks to the use of the notation Γ^n (Definition 4.1) and the fact that the AMPI scheme is itself much simpler than λ-policy iteration. Moreover, the results are deeper since we consider a possible error in the greedy step and more general concentration coefficients. Canbolat and Rothblum (2012) recently (and independently) developed an analysis of an approximate form of MPI. While Canbolat and Rothblum (2012) only consider the error in the greedy step, our work is more general since it takes into account both this error and the error in the value update. Note that it is required to consider both sources of error for the analysis of CBMPI. Moreover, Canbolat and Rothblum (2012) provide bounds when the errors are controlled in max-norm, while we consider the more general ℓ_p-norm. At a more technical level, Theorem 2 in Canbolat and Rothblum (2012) bounds the norm of the distance $v^* - v_k$ while we bound the loss $v^* - v^{\pi_k}$. Finally, if we derive a bound on the loss (using e.g., Theorem 1 in Canbolat and Rothblum 2012), this leads to a bound on the loss that is looser than ours. In particular, this does not allow to recover the standard bounds for AVI/API, as we managed to obtain here (c.f., Remark 4.3).

Remark 4.5. We can balance the influence of the concentrability coefficients (the bigger the q, the higher the influence) and the difficulty of controlling the errors (the bigger the q', the greater the difficulty in controlling the $\ell_{pq'}$-norms) by tuning the parameters q and q', given the condition that $\frac{1}{q}+\frac{1}{q'}=1$. This potential leverage is an improvement over the existing bounds and concentrability results that only consider specific values of these two parameters: $q=\infty$ and $q'=1$ in Munos (2007) and Munos and Szepesvári (2008), and $q=q'=2$ in Farahmand et al. (2010).

Remark 4.6. Interestingly, our loss bound for AMPI does not "directly" depend on m (although as we will discuss in the next section, it actually does depend "indirectly" through ϵ_k). For CBMPI, the parameter m controls the influence of the value function approximator, cancelling it out in the limit when m tends to infinity (see Equation 4.21). Assuming a fixed budget of sample transitions, increasing m reduces the number of rollouts used by the classifier and thus worsens its quality. In such a situation, m allows to make a trade-off between the estimation errors of the classifier and the overall value function approximation.

Remark 4.7. The result that we have just stated means the following: if one can control the errors ϵ_k and ϵ'_k, then the performance loss is also controlled. The main limitation of this result is that in general, even if we consider that there is no sampling noise ($N = \infty$ for all algorithms and $M = \infty$ for AMPI-V), the error ϵ_k of the *evaluation step* may grow arbitrarily and can make the algorithm diverge. The fundamental reason is that the composition of the approximation and the Bellman operator T_π is not necessarily contracting. A simple well-known pathological example is due to Tsitsiklis and Van Roy (1997) and involves a two-state uncontrolled MDP and a linear projection on a 1-dimensional space (that contains the real value function). Increasing the parameter m of the algorithm makes the operator T_π more contracting and in principle can address this issue. For instance, if we consider that we have a state space of finite size $|\mathcal{S}|$, and take the uniform distribution μ, it can be easily seen that for any v and v', we have

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

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

4 Finite-Sample Analysis of the Algorithms

In this section, we first show how the error terms ϵ_k and ϵ'_k appeared in Theorem 4.1 (Equations 4.20 and 4.21) are bounded in each of the three proposed algorithms, and then use the obtained results and derive finite-sample performance bounds for these algorithms. We first bound the *evaluation step error* ϵ_k. In AMPI-V and CBMPI, the evaluation step at each iteration k is a regression problem with the target $(T_{\pi_k})^m v_{k-1}$ and a training set of the form $\left\{\left(s^{(i)}, \widehat{v}_k(s^{(i)})\right)\right\}_{i=1}^N$ in which the states $s^{(i)}$ are i.i.d. samples from the distribution μ and $\widehat{v}_k(s^{(i)})$'s are unbiased estimates of the target computed using Equation 4.2. The situation is the same for AMPI-Q, except everything is in terms of action-value function Q_k instead of value function v_k. Therefore, in the following we only show how to bound ϵ_k in AMPI-V and CBMPI, the extension to AMPI-Q is straightforward.

We may use different function spaces $\mathcal{F}$ (linear or non-linear) to approximate the value function. Here we consider a linear architecture with parameters $\alpha \in \mathbb{R}^d$ and bounded (by L) basis functions $\{\varphi_j\}_{j=1}^d$, $\|\varphi_j\|_\infty \le L$ as detailed in Section 1.3.1 of Chapter 2. Now if we define v_k as the truncation (by $V_{\max}$) of the solution of the above linear regression problem, we may bound the *evaluation step error* ϵ_k using the following lemma.

Lemma 4.4 (Evaluation step error). *Consider the linear regression setting described above, then we have*

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

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

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

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

Proof. See Appendix E. □

After we showed how to bound the *evaluation step error* ϵ_k for the proposed algorithms, we now turn our attention to bounding the *greedy step error* ϵ'_k, that contrary to the evaluation step error, varies more significantly across the algorithms. While the greedy step error equals to zero in AMPI-Q, it is based on sampling in AMPI-V, and depends on a classifier in CBMPI. We first consider the simpler case of AMPI-V.

For any $\epsilon > 0$ and state s, when an action is identified as approximately optimal through Equation 4.1, the distance between the estimates $\frac{1}{M}\sum_{j=1}^M r_a^{(j)} + \gamma v_k(s_a^{(j)})$ for all actions a and their expected value $\mathbb{E}[r(s_0,a)+\gamma v_k(s_1)|s_0=s, a_0=a]$ can be proved to be smaller than $\frac{\epsilon}{2}$ with high-probability through Hoeffding's inequality, which implies that the error $\epsilon'_k(s)$ is bounded by ϵ. In AMPI-V, one needs to estimate approximately optimal action along several rollouts, which leads to the following bound.

Lemma 4.5 (Greedy step error of AMPI-V). *The greedy step error* ϵ'_k *in the AMPI-V algorithm is bounded at each state* $s\in\mathcal{S}$, *with probability at least* $1-\delta$, *as*

$$|\epsilon'_k(s)| \le 2\gamma V_{\max}\sqrt{\frac{2\log(2|\mathcal{A}|/\delta)}{M}} = e'_0(M,\delta).$$

Proof. See Appendix F. □

We now show how to bound ϵ'_k in CBMPI. From the definitions of ϵ'_k (Equation 4.17) and $\mathcal{L}_k^\Pi(\mu;\pi)$ (Equation 4.12), it is easy to see that $\|\epsilon'_k\|_{1,\mu} = \mathcal{L}_{k-1}^\Pi(\mu;\pi_k)$. This is because

$$\begin{aligned}\epsilon'_k(s) &= \max_{a\in\mathcal{A}} \Big[T_a(T_{\pi_{k-1}})^m v_{k-2}\Big](s) - \Big[T_{\pi_k}(T_{\pi_{k-1}})^m v_{k-2}\Big](s) && \text{(see Equation 4.10)}\\ &= \max_{a\in\mathcal{A}} Q_{k-1}(s,a) - Q_{k-1}\Big(s,\pi_k(s)\Big) = \mathcal{L}_{k-1}^\Pi(\mu;\pi_k). && \text{(see Equations 4.11 and 4.12)}\end{aligned}$$

Lemma 4.6 (Greedy step error of CBMPI). *Let Π be a policy space with finite VC-dimension $h = VC(\Pi)$ and μ be a distribution over the state space $\mathcal{S}$. Let N' be the number of states in $\mathcal{D}'_{k-1}$ drawn i.i.d. from μ, M be the number of rollouts per state-action pair used in the estimation of $\widehat{Q}_{k-1}$, and $\pi_k \in \operatorname{argmin}_{\pi\in\Pi} \widehat{\mathcal{L}}^{\Pi}_{k-1}(\widehat{\mu}, \pi)$ be the policy computed at iteration $k-1$ of CBMPI. Then, for any $\delta > 0$, we have*

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

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

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

Proof. See Appendix G. □

From Lemma 4.4, we have a bound on $\|\epsilon_k\|_{2,\mu}$ for all the three algorithms. Since $\|\epsilon_k\|_{1,\mu} \le \|\epsilon_k\|_{2,\mu}$, we also have a bound on $\|\epsilon_k\|_{1,\mu}$ for all the algorithms. On the other hand, from Lemma 4.6 , we have a bound on $\|\epsilon'_k\|_{1,\mu}$ for the CMBPI algorithm. This means that for CBMPI, we can control the RHS of Equation 4.21 in ℓ_1-norm, which in the context of Theorem 4.1 means $p = 1$, $q' = 1$, and $q = \infty$. Since ϵ'_k is zero for AMPI-Q and can be bounded point-wise for AMPI-V (Lemma 4.5), we can control the RHS of Equation 4.20 in both ℓ_1 and ℓ_2-norms. Controlling the RHS of Equation 4.20 in ℓ_1-norm means $p = 1$, $q' = 1$, and $q = \infty$ in the context of Theorem 4.1, while controlling it in ℓ_2-norm means either $p = 2$, $q' = 1$, and $q = \infty$, or $p = 1$, $q' = 2$, and $q = 2$. This leads to the main result of this section, finite sample performance bounds for the three proposed algorithms.

Theorem 4.2. *Let*

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

where $\mathcal{F}$ is the function space used by the algorithms and Π is the policy space used by CBMPI. With the notations of Theorem 4.1 and Lemmas 4.4-4.6, after k iterations, and with probability $1-\delta$, the expected loss $\mathbb{E}_\mu[l_k] = \|l_k\|_{1,\mu}$ of the proposed AMPI algorithms satisfy:[6]

AMPI-V:
$$\|l_k\|_{1,\mu} \le \frac{2(\gamma-\gamma^k)\mathcal{C}^{1,k,0}_{\infty}}{(1-\gamma)^2}\left(d_m + e_1(N, \frac{\delta}{k}) + e_2(N, \frac{\delta}{k})\right) + \frac{(1-\gamma^k)\mathcal{C}^{0,k,0}_{\infty}}{(1-\gamma)^2} e'_0(M, \frac{\delta}{k}) + g(k),$$

[6]As mentioned above, the bounds of AMPI-V and AMPI-Q may also be written with $(p = 2, q' = 1, q = \infty)$, and $(p = 1, q' = 2, q = 2)$.

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

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

Remark 4.8. The CBMPI bound in Theorem 4.2 allows us to restate Remark 4.6. Assume that we have a fixed budget $B = Nm + N'M|\mathcal{A}|(m+1)$ that we equally divide over the classifier and regressor. Note that the budget is measured in terms of the number of call to the generative model. Then up to constants and logarithmic factors, the bound has the form

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

This shows a trade-off in tuning the parameter m: a large value of m makes the influence of the regressor's error (both approximation and estimation errors) smaller, but at the same time makes the influence of the estimation error of the classifier bigger, in the final error. Note that we recover here the same trade-off as the one discussed for DPI-Critic in Remark 3.6 of Chapter 3 to which is added the fact the number of states in $\mathcal{D}$, N, also decreases with m, $N = O(B/m)$.

5 Experiments

For our experiments, we evaluate CBMPI in two different domains: **1)** the *Mountain Car* problem and **2)** the more challenging game of Tetris. In several experiments, we compare the performance of CBMPI with the DPI algorithm (Lazaric et al., 2010a), which is basically CBMPI without value function approximation. Hence, comparing DPI and CBMPI allows us to highlight the role of the value function approximation.

As discussed in Remark 4.6, the parameter m in CBMPI balances between the errors in evaluating the value function and the policy. We follow in this Section the same methodology as the one followed in the experiments run for DPI-Critic in Section 5 of Chapter 3. As discussed in Remark 3.6 of Chapter 3, one of the objectives of our experiments is to show the role of the parameters M, N, N', and m in the performance of CBMPI. However, since we almost always obtained our best results with $M = 1$, we only focus on the parameters m and N in our experiments. Moreover, as mentioned in Footnote 3, we implement a more sample efficient version of CBMPI by reusing the rollouts generated for the classifier in the regressor. More precisely, at each iteration k, for each state $s^{(i)} \in \mathcal{D}_k'$ and each action $a \in \mathcal{A}$, we generate one rollout of length $m+1$, i.e., $\left(s^{(i)}, a, r_0^{(i)}, s_1^{(i)}, a_1^{(i)}, \dots, a_m^{(i)}, r_m^{(i)}, s_{m+1}^{(i)}\right)$. We then take the rollout of action $\pi_k(s^{(i)})$, select its last m steps, i.e., $\left(s_1^{(i)}, a_1^{(i)}, \dots, a_m^{(i)}, r_m^{(i)}, s_{m+1}^{(i)}\right)$ (note that all the actions here have been taken according to the current policy π_k), use it to estimate the value function $\widehat{v}_k(s_1^{(i)})$, and add it to the training set of the regressor. This process guarantees to have $N = N'$.

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

5.1 Mountain Car

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

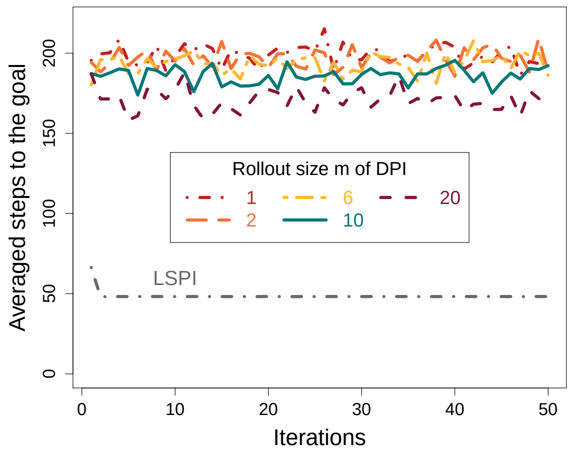
5.1.1 Problem Setting

Mountain Car is the problem of driving a car up to the top of a one-dimensional hill (see Figure 3.2). In MC, the setting is similar to the one already described in Section 5.1. We only report in the following some implementation elements that differ from Section 5.1. Here, the range of the uniform noise added to the actions is $[-0.2, 0.2]$. We recall also that each state s consists of the pair $(x, \dot{x})$ where x is the position of the car and $\dot{x}$ is its velocity.

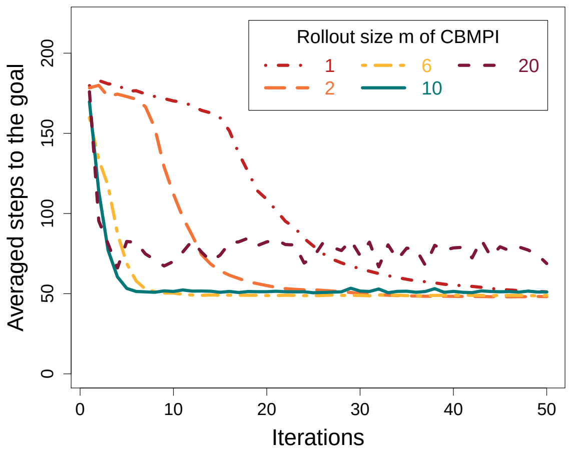
The policy space Π (classifier) is defined by a regularized support vector classifier (C-SVC) using the LIBSVM implementation by Chang and Lin (2011). We use the RBF kernel $\exp(-|u-v|^2)$ and set the cost parameter $C = 1000$. As discussed in Section 5.1, we again minimize the classification error instead of directly solving the cost-sensitive multi-class classification step as in Figure 4.3.

In our MC experiments, the policies learned by the algorithms are evaluated by the number of steps-to-go (average number of steps to reach the goal with a maximum of 300) averaged over $4,000$ independent trials. More precisely, we define the possible starting configurations (positions and velocities) of the car by placing a 20×20 uniform grid over the state space, and run the policy 10 times from each possible initial configuration. The performance of each algorithm is represented by a learning curve whose value at each iteration is the average number of steps-to-go of the policies learned by the algorithm at that iteration in 1000 separate runs of the algorithm.

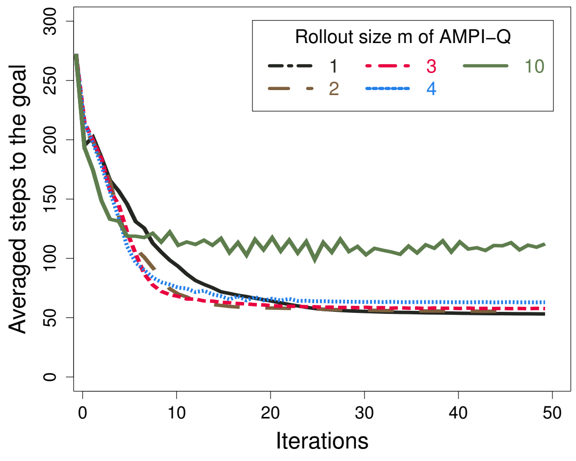
We tested the performance of DPI, CBMPI, and AMPI-Q on a wide range of parameters (m, M, N), but only report their performance for the best choice of M (as mentioned earlier, $M = 1$ was the best choice in all the experiments) and different values of m.



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

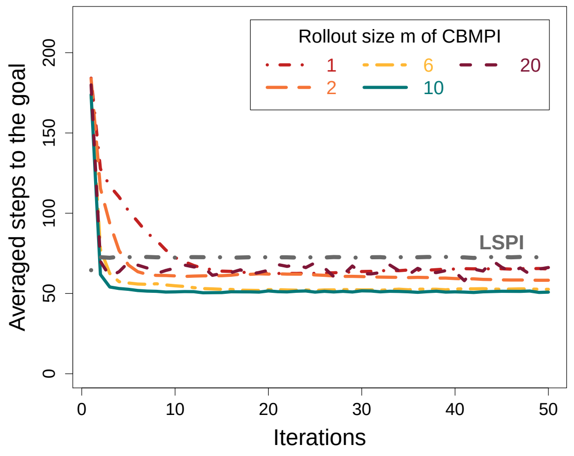


(b) Performance of CBMPI for different values of m.

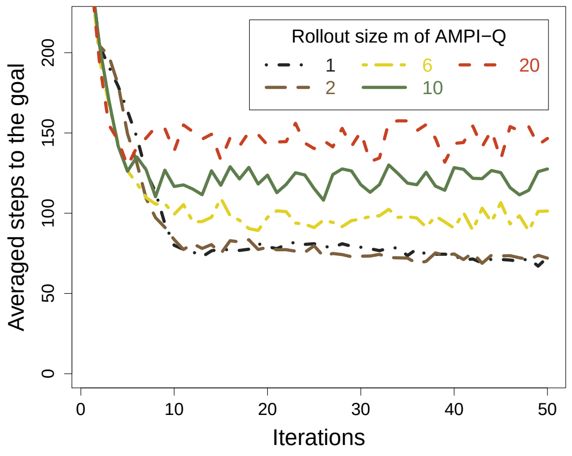


(c) Performance of AMPI-Q for different values of m.

Figure 4.4: Performance of the policies learned by **(a)** DPI and LSPI, **(b)** CBMPI, and **(c)** AMPI-Q algorithms in the Mountain Car (MC) problem, when we use a 3×3 RBF grid to approximate the value function. The results are averaged over $1,000$ runs. The total budget B is set to $4,000$ per iteration.



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



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

Figure 4.5: Performance of the policies learned by **(a)** CBMPI and LSPI and **(b)** AMPI-Q algorithms in the Mountain Car (MC) problem, when we use a 2×2 RBF grid to approximate the value function. The results are averaged over $1,000$ runs. The total budget B is set to $4,000$ per iteration.

5.1.2 Experimental Results

Figure 4.4 shows the learning curves of DPI, CBMPI, AMPI-Q, and LSPI algorithms with budget $B = 4,000$ per iteration and the function space $\mathcal{F}$ composed of a 3×3 RBF grid. We notice from the results that this space is rich enough to provide a good approximation for the value function components (e.g., in CBMPI, for $(T_\pi)^m v_{k-1}$ defined by Equation 4.16). Therefore, LSPI and DPI obtain the best and worst results with about 50 and 160 steps to reach the goal, respectively. The best DPI results are obtained with the large value of $m = 20$. DPI performs better for large values of m because the reward function is constant everywhere except at the goal, and thus, a DPI rollout is only *informative* if it reaches there. We also report the performance of CBMPI and AMPI-Q for different values of m. The value function approximation is so accurate that CBMPI and AMPI-Q achieve performance similar to LSPI for $m < 20$. However when m is large ($m = 20$), the performance of these algorithms is worse, because in this case, the rollout set does not have enough elements (N small) to learn the greedy policy and value function well. Note that as we increase m (up to $m = 10$), CBMPI and AMPI-Q converge faster to a good policy.

Although this experiment shows that the use of a critic in CBMPI compensates for the truncation of the rollouts (CBMPI performs better than DPI), most of this advantage is due to the richness of the function space $\mathcal{F}$ (LSPI and AMPI-Q perform as well as CBMPI – LSPI even converges faster). Therefore, it seems that it would be more efficient to use LSPI instead of CBMPI in this case. In the next experiment, we study the performance of these algorithms when the function space $\mathcal{F}$ is less rich, composed of a 2×2 RBF grid. The results are reported in Figure 4.5. Now, the

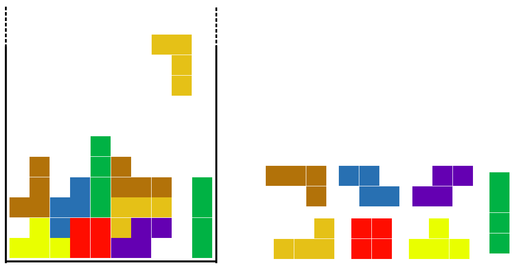


Figure 4.6: A screen-shot of the game of Tetris and the seven pieces (shapes) used in the game.

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

Finally, by comparing the results of CBMPI on the 2×2 and 3×3 RBF grids, one can observe that CBMPI converges faster when the RBF grid is smaller. This is due to the fact that the RBF in the 2×2 case are more spread out that in the 3×3 case which permits the value function estimates in the value iteration procedure to propagate more quickly.

5.2 Tetris

Tetris is a popular video game created by Alexey Pajitnov in 1985. The game is played on a grid originally composed of 20 rows and 10 columns, where pieces of 7 different shapes fall from the top (see Figure 4.6). The player has to choose where to place each falling piece by moving it horizontally and rotating it. When a row is filled, it is removed and all the cells above it move one line down. The goal is to remove as many rows as possible before the game is over, i.e., when there is no space available at the top of the grid for the new piece. Here, we consider the variation of the game in which the player knows only the current falling piece, and not the next several coming pieces. This game constitutes an interesting optimization benchmark in which the goal is to find a controller (policy) that maximizes the average (over multiple games) number of lines removed in a game (score).[7] This optimization problem is known to be computationally hard. It contains a huge number of board configurations (about $2^{200} \simeq 1.6 \times 10^{60}$), and even in the case that the sequence of pieces is known in advance, finding the strategy to maximize the score is an NP hard problem Demaine et al. (2003).

Approximate dynamic programming (ADP) and reinforcement learning (RL) algorithms have been used in Tetris. These algorithms formulate Tetris as a MDP in

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

which the state is defined by the current board configuration plus the falling piece, the actions are the possible orientations of the piece and the possible locations that it can be placed on the board,[8] and the reward is defined such that maximizing the expected sum of rewards from each state coincides with maximizing the score from that state. Since the state space is large in Tetris, these methods use value function approximation schemes (often linear approximation) and try to tune the value function parameters (weights) from game simulations. The first application of ADP in Tetris seems to be by Tsitsiklis and Van Roy (1996). They used the approximate value iteration algorithm with two state features: the board height and the number of holes in the board, and obtained a low score of 30 to 40. Bertsekas and Ioffe (1996) proposed the λ-Policy Iteration (λ-PI) algorithm (a generalization of value and policy iteration) and applied it to Tetris. They approximated the value function as a linear combination of a more elaborate set of 22 features and reported the score of $3,200$ lines averaged over 100 games. The exact same empirical study was revisited recently by Scherrer (2013), who corrected an implementation bug in Bertsekas and Ioffe (1996), and reported more stable learning curves and the score of $4,000$ lines. At least three other ADP and RL papers have used the same set of features, we call them the "Bertsekas features", in the game of Tetris. Kakade (2001) applied a natural policy gradient method to Tetris and reported a score of about $6,800$ lines, without specifying over how many games this number was averaged. Farias and Van Roy (2006) applied a linear programming algorithm to the game and achieved the score of $4,700$ lines averaged over 90 games. Furmston and Barber (2012) proposed an approximate Newton method to search in a policy space and were able to obtain a score of about $14,000$. However, since they have not reported the details of their experiments, it is not clear whether their result is statistically significant.

Despite all the above applications of ADP in Tetris (and possibly more), for a long time, the best Tetris controller was the one designed by Dellacherie (Fahey, 2003). He used a heuristic evaluation function to give a score to each possible strategy (in a way similar to value function in ADP), and eventually returned the one with the highest score. Dellacherie's evaluation function is made of 6 high-quality features with weights chosen by hand, and achieved a score of about $5,000,000$ lines (Thiéry and Scherrer, 2009a). Szita and Lőrincz (2006) used the "Bertsekas features" and optimized the weights by running a black box optimizer based on the cross entropy (CE) method (Rubinstein and Kroese, 2004). They reported the score of $350,000$ lines averaged over 30 games, outperforming the ADP and RL approaches that used the same features. More recently, Thiéry and Scherrer (2009b) selected a set of 9 features (including those of Dellacherie's) and optimized the weights with the CE method. This led to the best publicly known controller (to the best of our knowledge) with the score of around $35,000,000$ lines.

Due to the high variance of the score and its sensitivity to some implementation details (Thiéry and Scherrer, 2009a), it is difficult to have a precise evaluation of Tetris

[8] The total number of actions at a state depends on the shape of the falling piece, with the maximum of 34 actions in a state, i.e., $|\mathcal{A}| \leq 34$.

controllers. However, our brief tour d'horizon of the literature, and in particular the work by Szita and Lőrincz (2006) (optimizing the "Bertsekas features" by CE), indicate that ADP algorithms, even with relatively good features, have performed extremely worse than the methods that directly search in the space of policies (such as CE and genetic algorithms). It is important to note that most of these ADP methods are *value function based* algorithms that first define a value function representation (space) and then search in this space for a good function, which later gives us a policy.

The main motivation of our experiments comes from the above observation. This observation makes us conjecture that Tetris is a game whose policy space is easier to represent, and as a result to search in, than its value function space. Therefore, in order to obtain a good performance with ADP algorithms in this game, we should use those ADP methods that search in a policy space, instead of the more traditional ones that search in a value function space.

In this section, we put our conjecture to test by applying CBMPI to the game of Tetris, and compare its performance with the CE method and the λ-PI algorithm. The choice of CE and λ-PI is because the former has achieved the best known results in Tetris and the latter's performance is among the best reported for value function based ADP algorithms. Our extensive experimental results show that for the first time an ADP algorithm, namely CBMPI, obtains the best results reported in the literature for Tetris in both small 10×10 and large 10×20 boards. The CBMPI's results outperform those achieved by the CE method in the large board, and moreover CBMPI uses considerably fewer (almost 1/6) samples (call to the generative model of the game) than CE.

5.2.1 Algorithms and Experimental Setup

In this section, we briefly describe the algorithms used in our experiments: the cross entropy (CE) method, our particular implementation of CBMPI, and its slight variation DPI. We refer the readers to Scherrer (2013) for λ-PI. We begin by defining some terms and notations. A state s in Tetris consists of two components: the description of the board b and the type of the falling piece. All controllers rely on an evaluation function that gives a value to each possible action at a given state. Then, the controller chooses the action with the highest value. In ADP, algorithms aim at tuning the weights such that the evaluation function approximates well the optimal expected future score from each state. Since the total number of states is large in Tetris, the evaluation function f is usually defined as a linear combination of a set of features ϕ, i.e., $f(\cdot) = \phi(\cdot)\theta$[9]. We can think of the parameter vector θ as a policy (controller) whose performance

[9]Their exist alternatives to this use of a linear evalution function. For instance, every action to be taken can be computed using a Monte-Carlo tree search. This approach has been applied recently to the game of Tetris (Cai et al., 2011). While this is a promising line of research, the results reported in their paper are only comparing the number of games won over a benchmark policy rather than in terms of the averaged lines removed thus making it hard to compare to other policies. Moreover the Monte-Carlo approach is expected to require more computation at every step of the game compared to the evaluation of the linear evaluation function.

is specified by the corresponding evaluation function $f(\cdot) = \phi(\cdot)\theta$. The features used in Tetris for a state-action pair (s, a) may depend on the description of the board b' resulted from taking action a in state s, e.g., the maximum height of b'. Computing such features requires the knowledge of the game's dynamics, which is known in Tetris. We use the following sets of features, plus a constant offset feature, in our experiments:[10]

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

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

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

The Cross Entropy (CE) Method: CE (see Rubinstein and Kroese (2004)) is an iterative method whose goal is to optimize a function f parameterized by a vector $\theta \in \Theta$ by direct search in the parameter space Θ. This technique has been used to solve RL problems by directly optimizing the performance of a paramatrized policy (Mannor et al., 2003). For the case of the game of Tetris, Figure 4.7 contains the pseudo-code of the CE algorithm used in our experiments (Szita and Lőrincz, 2006, Thiéry and Scherrer, 2009b). At each iteration k, we sample n parameter vectors $\{\theta_i\}_{i=1}^n$ from a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2 I)$. At the beginning, the parameters of this Gaussian have been set to cover a wide region of Θ. For each parameter θ_i, we play G games and calculate the average number of rows removed by this controller (an estimate of the evaluation function). We then select $\lfloor \zeta n \rfloor$ of these parameters with the highest score, $\theta'_1, \ldots, \theta'_{\lfloor \zeta n \rfloor}$, and use them to update the mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}^2$ of the Gaussian distribution, as shown in Figure 4.7. This updated Gaussian is used to sample the n parameters at the next iteration. The goal of this update is to sample more parameters from the promising parts of Θ at the next iteration, and eventually converge to a global maximum of f. In our experiments, in the pseudo-code of Figure 4.7, we set $\zeta = 0.1$ and $\eta = 4$, the best parameters reported in Thiéry and Scherrer (2009b). We also set $n = 1{,}000$ and $G = 10$ in the small board and $n = 100$ and $G = 1$ in the large board.

[10]For a precise definition of the features, see Thiéry and Scherrer (2009a) or the documentation of their code (Thiéry and Scherrer, 2010b). Note that the constant offset feature has no incidence when modelling policies while it plays a role to approximate the value functions.

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

Figure 4.7: The pseudo-code of the cross-entropy (CE) method used in our experiments.

Our Implementation of CBMPI (& DPI): We use the algorithm whose pseudo-code is shown in Figure 4.3. We sample the rollout states from the trajectories generated by a very good policy for Tetris, namely the DU controller (Thiéry and Scherrer, 2009b) (the choice of the sampling distribution μ in Figure 4.3). Since the DU policy is good, this rollout set is biased towards boards with small height. We noticed from our experiments that the performance can be significantly improved if we use boards with different heights in the rollout sets. This means that better performance can be achieved with more uniform sampling distribution, which is consistent with what we can learn from the CBMPI and DPI performance bounds. We set the initial value function parameter to $\alpha = \bar{0}$ and select the initial policy π_1 (policy parameter β) randomly. We also set the CMA-ES parameters (classifier parameters) to $\zeta = 0.5$, $\eta = 0$, and n equal to 15 times the number of features. Finally, we set the discount factor $\gamma = 1$.

- **Regressor:** We use linear function approximation for the value function, i.e., $\widehat{v}_k(s^{(i)}) = \phi(s^{(i)})\alpha$, where $\phi(\cdot)$ and α are the feature and weight vectors, and minimize the empirical error $\widehat{\mathcal{L}}_k^{\mathcal{F}}(\widehat{\mu}; v)$ using the standard least-squares method.

- **Classifier:** The training set of the classifier is of size N with $s^{(i)} \in \mathcal{D}'_k$ as input and $\big(\max_a \widehat{Q}_k(s^{(i)}, a) - \widehat{Q}_k(s^{(i)}, a_1), \dots, \max_a \widehat{Q}_k(s^{(i)}, a) - \widehat{Q}_k(s^{(i)}, a_{|\mathcal{A}|})\big)$ as output. We use the policies of the form $\pi_\beta(s) \in \operatorname{argmax}_a \psi(s, a)\beta$, where ψ is the policy feature vector (possibly different from the value function feature vector ϕ) and $\beta \in \mathcal{B}$ is the policy parameter vector. Contrarily to the previous use of a surrogate classification error in the MC problem, we compute here the next policy π_{k+1} by directly minimizing the empirical error $\widehat{\mathcal{L}}_k^{\Pi}(\widehat{\mu}; \pi_\beta)$, defined by (4.14). To do so, we use the covariance matrix adaptation evolution strategy (CMA-ES) algorithm (Hansen and Ostermeier, 2001). In order to evaluate a policy $\beta \in \mathcal{B}$ in CMA-ES, we only need to compute $\widehat{\mathcal{L}}_k^{\Pi}(\widehat{\mu}; \pi_\beta)$, and given the training set, this procedure does not require any simulation of the game. This is in contrary with policy evaluation in CE that requires playing several games, and it is the main reason that we obtain the same performance as CE with CBMPI with 1/6 number of samples when learning on large (10×20) board.

Remark 4.9 (on the use of the cross entropy optimizer). In our implementation of the cross entropy optimizer for Tetris, the parameter space of the policies, Θ, is $\mathbb{R}^d$, where d is the number of features and thus also the size of the weight vector θ. However, a policy defined by its weight vector θ is exactly the same as the policy defined by the weight vector $\alpha\theta$ for any $\alpha \in \mathbb{R}$. Therefore, the parameter space could be restricted to $d-1$ dimensions. For instance one could adopt an hyper-spherical coordinate system for $\mathbb{R}^d$ where, as a consequence, the radial coordinate would have no influence on the policy. In fact, by fixing the radial coordinate to 1, the parameter space Θ would be defined as a $d-1$-sphere. Then, to adapt the CE method, we now need to sample from a Gaussian distribution defined over a $d-1$-sphere. This distribution is the Von Mises–Fisher distribution. We did not explore this technique in this work but conjecture that it could make the convergence of cross entropy quicker in the game of Tetris as it removes one dimension in the policy search task.

Goschin et al. (2013) showed that the standard version of CE that selects at each iteration, as explained above, the top ρN controllers is subject to select controllers that do not have the best expected performance but rather the ones whose performances have the highest quantile (most likely because of a large variance). Goschin et al. (2013) designed the Proportional CE method for a simplified version of Tetris that only considers the tetrominos with S or Z shapes where the new Gaussian distribution is fit using all the controllers tested $(\rho = 1)$ but weighting them proportionally to their performance. Proportional CE was showed to have better stability and superior expected performance that the standard CE in the simplified game of Tetris. Our experiments were conducted with the standard version of CE, and therefore it would be of interest to also compute the performance of Proportional CE in the original game of Tetris.

5.2.2 Experimental Results

In our Tetris experiments, the policies learned by the algorithms are evaluated by their score (average number of rows removed in a game started with an empty board) averaged over 200 games in the small 10×10 board and over 20 games in the large 10×20 board (since the game takes much more time to complete in the large board). The performance of each algorithm is represented by a learning curve whose value at each iteration is the average score of the policies learned by the algorithm at that iteration in 100 separate runs of the algorithm. The curves are wrapped in their confidence intervals that are computed as three times the standard deviation of the estimation of the performance at each iteration. In addition to their score, we also evaluate the algorithms by the number of samples they use. In particular, we show that CBMPI/DPI use 6 times less samples than CE in the large board. As discussed in Section 5.2.1, this is due the fact that although the classifier in CBMPI/DPI uses a direct search in the space of policies (for the greedy policy), it evaluates each candidate policy using the empirical error of Equation 4.14, and thus, does not require any simulation of the game (other than those used to estimate the $\widehat{Q}_k$'s in its training set). In fact, the budget B of CBMPI/DPI is fixed in advance by the number of rollouts NM and the rollout's length m as $B = (m+1)NM|\mathcal{A}|$. In contrary, CE evaluates a candidate policy by playing several games, a process that can be extremely costly (sample-wise), especially for good policies in the large board.

We first run the algorithms on the small board to study the role of their parameters and to select the best features and parameters, and then use the selected features and parameters and apply the algorithms to the large board. Finally, we compare the best policies found in our experiments with the best controllers reported in the literature (Tables 4.1 and 4.2).

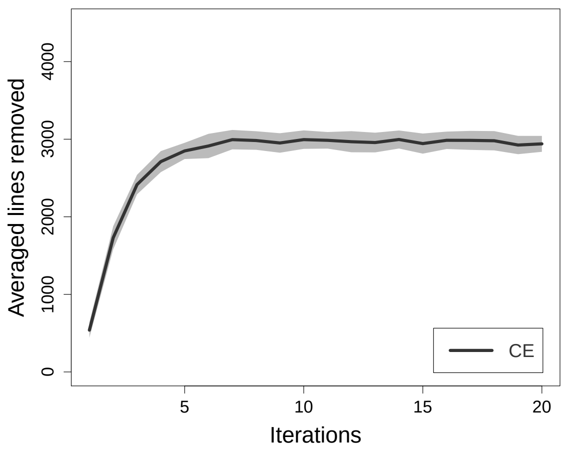
Small (10 × 10) Board

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

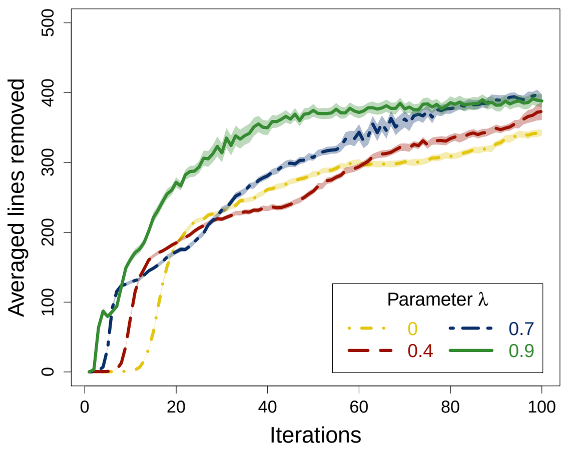
D-T Features: Figure 4.8 shows the learning curves of CE, λ-PI, DPI, and CBMPI algorithms. Here we use D-T features plus constant offset for the evaluation function in CE, the value function in λ-PI, and the policy in DPI and CBMPI. We ran CBMPI with different choices of features for the value function and "D-T plus Bertsekas plus the 5 RBF features and constant offset" achieved the best performance (Figure 4.8(d)).[11] The budget of CBMPI and DPI is set to $B = 8,000,000$ per iteration. The CE method reaches the score 3000 after 10 iterations using an average budget $B = 65,000,000$. λ-PI with the best value of λ only manages to score 400. In Figure 4.8(c), we report the performance of DPI for different values of m. DPI achieves its best performance for $m = 5$ and $m = 10$ by removing $3,400$ lines on average. As explained in Section 5.1, having short rollouts ($m = 1$) in DPI leads to poor action-value estimates $\widehat{Q}$, while having too long rollouts ($m = 20$) decreases the size of the training set of the classifier N. CBMPI outperforms the other algorithms, including CE, by reaching the score of $4,300$ for $m = 2$. This value of $m = 2$ corresponds to $N = \frac{8000000}{(2+1)\times 34} \approx 78,000$. Note that unlike DPI, CBMPI achieves good performance with very short rollouts $m = 1$. This indicates that CBMPI is able to approximate the value function well, and as a result, build a more accurate training set for its classifier than DPI. However, used directly as controllers, those value functions only remove around 400 lines. This suggests that the D-T features are more suitable to represent the policies than the value functions in Tetris.

This also suggests that the difference in term of performance between CBMPI and the standard *value function based ADP* methods as λ-PI is in how the greedy policy is

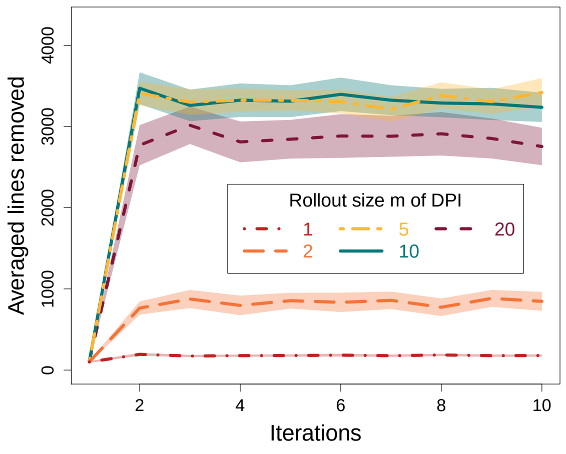
[11]Note that we use D-T+5 features only for the value function of CBMPI, and thus, we have a fair comparison between CBMPI and DPI. To have a fair comparison with λ-PI, we ran this algorithm with D-T+5 features, and it only raised its performance to 800, still far from the CBMPI's performance.



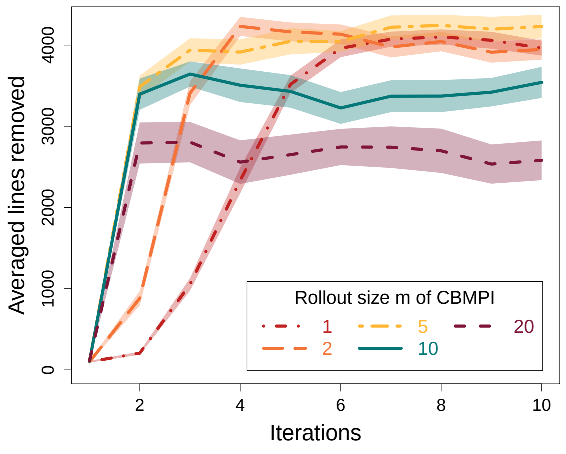
(a) The cross-entropy (CE) method.



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



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



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

Figure 4.8: Learning curves and confidence intervals of CE, λ-PI, DPI, and CBMPI algorithms using the 9 Dellacherie-Thiery (D-T) features on the small 10×10 board. The results are averaged over 100 runs of the algorithms.

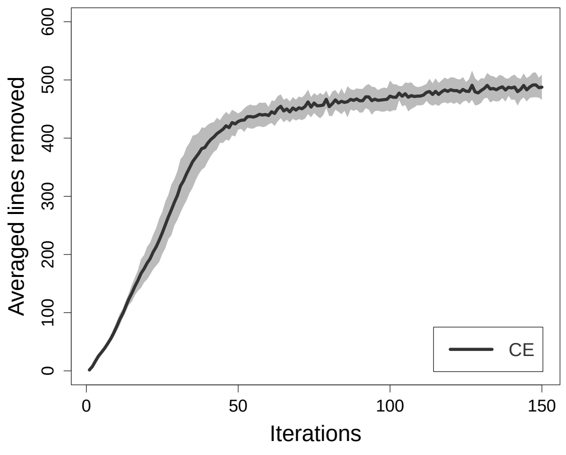
computed. Specifically, at each iteration CBMPI algorithms obtain the entire greedy policy as the output of a classifier which uses all the action-value function estimates over the entire rollout set, while in the standard methods, at every given state, the required action from the greedy policy is individually calculated based on some local values given by the approximation of the value function of the current policy.

The results of Figure 4.8 show that an ADP algorithm, namely CBMPI, outperforms the CE method using a similar budget (80 vs. 65 millions after 10 iterations). Note that CBMPI takes less iterations to converge than CE. More generally Figure 4.8 confirms the superiority of the policy search and classification-based PI methods to value function based ADP algorithms (λ-PI). Despite this improvement, the good results obtained by DPI in Tetris indicate that with small rollout horizons like $m = 5$, one has already fairly accurate action value estimates in order to detect greedy actions accurately (at each iteration).

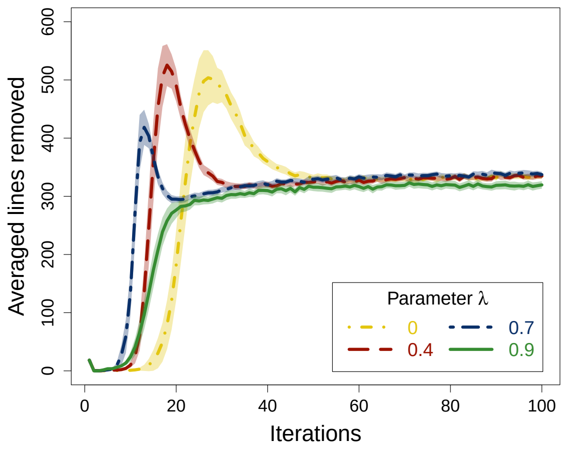
Bertsekas Features: Figure 4.9(a)-(c) show the performance of CE, λ-PI, DPI, and CBMPI algorithms. Here all the approximations in the algorithms are with the Bertsekas features plus constant offset. CE achieves the score 500 after about 60 iterations and outperforms λ-PI with score 350. It is clear that the Bertsekas features lead to much weaker results than those obtained by the D-T features (Figure 4.8) for all the algorithms. We may conclude then that the D-T features are more suitable than the Bertsekas features to represent both value functions and policies in Tetris. In DPI and CBMPI, we managed to obtain results similar to CE, only after multiplying the per iteration budget B used in the D-T experiments by 10. Indeed, CBMPI and DPI need more samples to solve the classification and regression problems in this 22-dimensions weight vector space than with the 9 D-T features. Moreover, in the classifier, the minimization of the empirical error through the CE method (see Equation 4.9) was converging most of the times to a local minimum. To solve this issue, we run multiple times the minimization problem with different starting points and small initial covariance matrices for the Gaussian distribution in order to force local exploration of different parts of the weight vector areas. However, CBMPI and CE use the same number of samples, $150,000,000$, when they converge after 2 and 60 iterations, respectively (see Figure 4.9). Note that DPI and CBMPI obtain the same performance, which means that the use of a value function approximation by CBMPI does not lead to a significant performance improvement over DPI. At the end, we tried several values of m in this setting among which $m = 10$ achieved the best performance for both DPI and CBMPI.

Large (10×20) Board

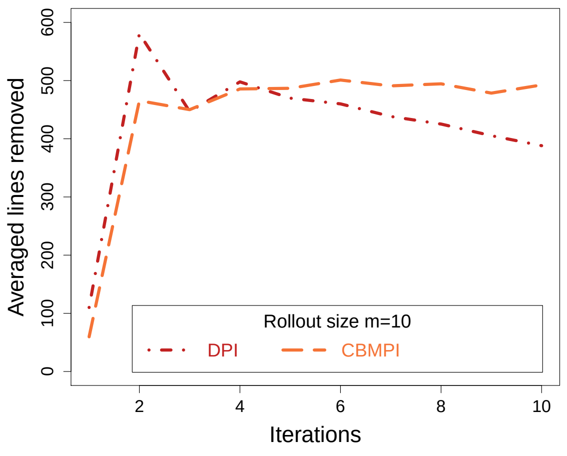
We now use the best parameters and features in the small board experiments, run CE, DPI, and CBMPI algorithms in the large board, and report their results in Figure 4.10(left). We also report the results of λ-PI in the large board in Figure 4.10(right). The per iteration budget of DPI and CBMPI is set to $B = 32,000,000$. While λ-PI with per iteration budget $620,000$, at its best, achieves the score of $2,500$, DPI and CBMPI, with $m = 10$, reach the scores of $12,000,000$ and $27,000,000$ after



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



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



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

Figure 4.9: (a)-(c) Learning curves and confidence intervals of CE, λ-PI, DPI, and CBMPI algorithms using the 22 Bertsekas features on the small 10×10 board.

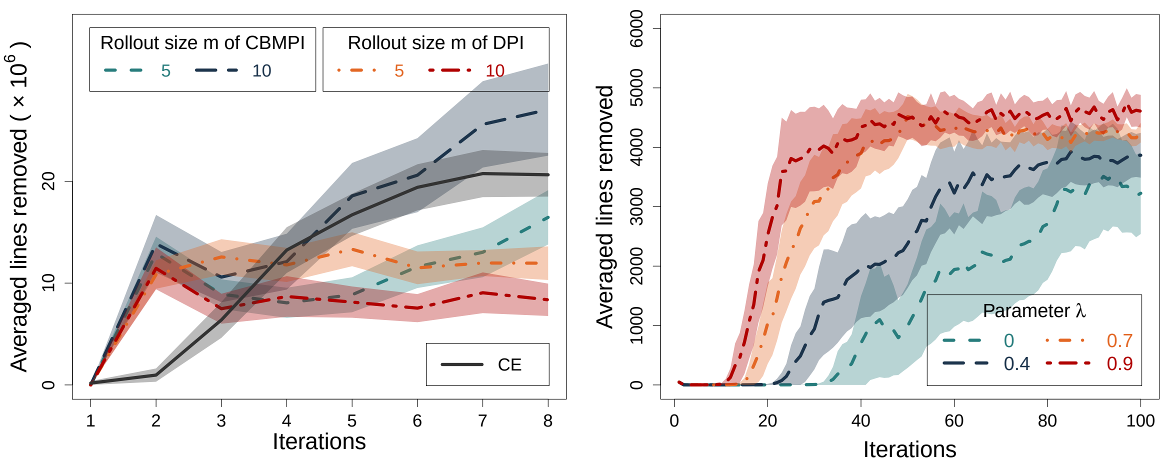


Figure 4.10: Learning curves and confidence intervals of CBMPI, DPI and CE (left), and λ-PI (right) using the 9 features listed in Table 4.2 on the large 10×20 board. The total budget B of CBMPI and DPI is set to 32,000,000 per iteration.

3 and 7 iterations, respectively. CE does not match the performances of CBMPI with the score of $20,000,000$ after 8 iterations, moreover this is achieved with almost 6 times more samples: after 8 iterations, CBMPI and CE use $256,000,000$ and $1,700,000,000$ samples, respectively.

Comparison of the best policies

So far the reported scores for each algorithm was averaged over the policies learned in 100 separate runs. Here we select the best policies observed in our all experiments and compute their scores more accurately by averaging over $10,000$ games. We then compare these results with the best policies reported in the literature, i.e., DU and BDU (Thiéry and Scherrer, 2009b) in both small and large boards in Table 4.1. The DT-10 and DT-20 policies, whose weights and features are given in Table 4.2[12], are policies learned by CBMPI with D-T features in the small and large boards, respectively. As shown in Table 4.1, DT-10 removes $5,000$ lines and outperforms DU, BDU, and DT-20 in the small board. Note that DT-10 is the only policy among these four that has been learned in the small board. In the large board, DT-20 obtains the score of $51,000,000$ and not only outperforms the other three policies, but also achieves the best reported result in the literature (to the best of our knowledge).

Note that learning with the 10×10 board and the 10×20 board are two different problems as DT-10 outperforms DT-20 in the former while it is the opposite in the

[12]Note that in the standard code by Thiéry and Scherrer (2010b), there exist two versions of the feature "board wells" numbered 6 and -6. In our experiments, we used both versions of the feature. The -6 feature was used in CBMPI in Gabillon et al. (2013) as it is the more computationally efficient of the two. Here it is still the one with which we found the DT-10 and the DT 20. The 6 feature was used in Gabillon et al. (2013) for CE and is now used also here for most of the reported curves for CBMPI. Indeed, we realize that better performance could be obtained with the 6 feature and that our previous comparison, in Gabillon et al. (2013), were in favour of CE.

latter.

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

Table 4.1: Average (over 10,000 games) score of DU, BDU, DT-10, and DT-20 policies. The 99%-confidence intervals around these average scores have radiuses equal to 3% of the average scores.

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

Table 4.2: The weights of the 9 Dellacherie-Thiery features in DT-10 (left) and DT-20 (right) policies.

6 Conclusions and Extensions

In this chapter, we studied a dynamic programming algorithm, called modified policy iteration (MPI), that despite its generality that contains the celebrated policy and value iteration methods, has not been thoroughly investigated in the literature. We proposed three approximate MPI (AMPI) algorithms that are extensions of the well-known ADP algorithms: fitted-value iteration, fitted-Q iteration, and classification-based policy iteration. Interestingly, the classification-based implementation of AMPI permits to give a new version and a broader framework to the DPI-Critic algorithm introduced in Chapter 3. We reported an error propagation analysis for AMPI that unifies those for approximate policy and value iteration. We also provided a finite-sample analysis for the classification-based implementation of AMPI (CBMPI), whose analysis is more general than the other presented AMPI methods. Our results indicate that the parameter of MPI allows us to control the balance of errors (in value function approximation and estimation of the greedy policy) in the final performance of CBMPI. The role of this parameter is illustrated in extensive experiments.

Remarkably, in the game of Tetris, a game that has always been challenging for approximate dynamic programming (ADP), our results showed that for the first time an ADP algorithm (CBMPI) performed extremely well in both small 10×10 and large 10×20 boards. Previously, the cross entropy (CE) methods which are much simpler black box optimization methods, had surprisingly produced controllers far superior to those learned by the ADP algorithms. We showed that CBMPI, a policy-based

ADP, achieved better performance, even with considerably fewer samples in the large board, than the state-of-the-art CE methods. In particular, the best policy learned by CBMPI obtained the performance of $51,000,000$ lines on average, a new record in the large board of Tetris. In the Tetris literature, a third policy-based approach has been used, namely the policy gradient algorithms. A closer look reveals that their poor results are mostly due to their use of features known to be bad to represent the policies (the Bertsekas features). To have a more complete comparison, those algorithms, and especially the recent approach by Furmston and Barber (2012), should be tested using the Dellacherie features. On the CBMPI side, plans to improve the results include making the policy updates asynchronous (to speed-up the learning process), and considering larger policy spaces with possibly stochastic and/or non-stationary policies in a similar fashion to Scherrer and Lesner (2012). Finally, this work leaves open the question of understanding why value function-based ADP algorithms fail at the game of Tetris. We conjecture that this could be addressed by going beyond the standard linear architecture of the value function and leave it as an interesting future work.

Also for future work, extension of CBMPI to problems with continuous action space is an interesting direction to pursue. In another direction, our empirical results in Tetris have given yet another evidence in showing how critical the choice of the rollout set distribution can be. In the line of the work by Rexakis and Lagoudakis (2012), more effort should be put in designing algorithms where a “good” rollout set distribution is learned. In fact, we tried to address the related problem of allocating non uniformly the rollouts over the states of the rollout set and the actions in $\mathcal{A}$ to improve the performance of the classification-based policy iteration algorithms. This actually motivated the bandit problem studied in the following chapter (see also the related paragraph in Section 5.1 of Chapter 5).

To continue with future work, we might also consider different and more general implementations of the cost-sensitive classifier to learn the policies. Indeed in our experiments so far, we either replaced the cost-sensitive classifier by a classifier minimizing a 0-1 loss or, in the specific case of the game of Tetris, we cast the classification problem as an optimization problem where policies are not learnt directly but through evaluation functions. The design of a multi-class cost-sensitive classifier is an open problem for most of the standard classification technique as SVM (Masnadi-Shirazi et al., 2012), Multiboost (Benbouzid et al., 2012) etc... Yet, recent advances due to Pires et al. (2013) might give us the necessary new tools to implement a general and efficient cost-sensitive classifier.

Appendix

In the following appendices, the proof of Lemmas 4.1 to 4.6 and Theorem 4.1 are provided.

A Proof of Lemma 4.1

Before we start, we recall the following definitions:

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

Bounding b_k

$$\begin{aligned} b_k &= v_k - T_{\pi_{k+1}} v_k = v_k - T_{\pi_k} v_k + T_{\pi_k} v_k - T_{\pi_{k+1}} v_k \overset{(a)}{\leq} v_k - T_{\pi_k} v_k + \epsilon'_{k+1} \\ &= v_k - \epsilon_k - T_{\pi_k} v_k + \gamma P_{\pi_k} \epsilon_k + \epsilon_k - \gamma P_{\pi_k} \epsilon_k + \epsilon'_{k+1} \\ &\overset{(b)}{=} v_k - \epsilon_k - T_{\pi_k}(v_k - \epsilon_k) + (I - \gamma P_{\pi_k})\epsilon_k + \epsilon'_{k+1}. \end{aligned} \tag{4.23}$$

Using the definition of x_k, i.e.,

$$x_k \triangleq (I - \gamma P_{\pi_k})\epsilon_k + \epsilon'_{k+1}, \tag{4.24}$$

we may write Equation 4.23 as

$$\begin{aligned} b_k &\leq v_k - \epsilon_k - T_{\pi_k}(v_k - \epsilon_k) + x_k \overset{(c)}{=} (T_{\pi_k})^m v_{k-1} - T_{\pi_k}(T_{\pi_k})^m v_{k-1} + x_k \\ &= (T_{\pi_k})^m v_{k-1} - (T_{\pi_k})^m (T_{\pi_k} v_{k-1}) + x_k \\ &= (\gamma P_{\pi_k})^m (v_{k-1} - T_{\pi_k} v_{k-1}) + x_k = (\gamma P_{\pi_k})^m b_{k-1} + x_k. \end{aligned} \tag{4.25}$$

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

Bounding d_k

$$\begin{aligned} d_{k+1} &= v^* - (T_{\pi_{k+1}})^m v_k = T_{\pi^*} v^* - T_{\pi^*} v_k + T_{\pi^*} v_k - T_{\pi_{k+1}} v_k + T_{\pi_{k+1}} v_k - (T_{\pi_{k+1}})^m v_k \\ &\overset{(a)}{\leq} \gamma P_{\pi^*}(v^* - v_k) + \epsilon'_{k+1} + g_{k+1} = \gamma P_{\pi^*}(v^* - v_k) + \gamma P_{\pi^*}\epsilon_k - \gamma P_{\pi^*}\epsilon_k + \epsilon'_{k+1} + g_{k+1} \\ &\overset{(b)}{=} \gamma P_{\pi^*}\Big(v^* - (v_k - \epsilon_k)\Big) + y_k + g_{k+1} = \gamma P_{\pi^*} d_k + y_k + g_{k+1} \\ &\overset{(c)}{=} \gamma P_{\pi^*} d_k + y_k + \sum_{j=1}^{m-1} (\gamma P_{\pi_{k+1}})^j b_k. \end{aligned} \tag{4.26}$$

(a) This step is from the definition of ϵ'_{k+1} (see step **(a)** in bounding b_k) and by defining g_{k+1} as follows:

$$g_{k+1} \triangleq T_{\pi_{k+1}} v_k - (T_{\pi_{k+1}})^m v_k. \tag{4.27}$$

(b) This is from the definition of y_k, i.e.,

$$y_k \triangleq -\gamma P_{\pi^*} \epsilon_k + \epsilon'_{k+1}. \tag{4.28}$$

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

$$g_{k+1} = T_{\pi_{k+1}} v_k - (T_{\pi_{k+1}})^m v_k = \sum_{j=1}^{m-1} \Big[(T_{\pi_{k+1}})^j v_k - (T_{\pi_{k+1}})^{j+1} v_k \Big] \tag{4.29}$$

$$= \sum_{j=1}^{m-1} \Big[(T_{\pi_{k+1}})^j v_k - (T_{\pi_{k+1}})^j (T_{\pi_{k+1}} v_k) \Big] = \sum_{j=1}^{m-1} (\gamma P_{\pi_{k+1}})^j (v_k - T_{\pi_{k+1}} v_k) \tag{4.30}$$

$$= \sum_{j=1}^{m-1} (\gamma P_{\pi_{k+1}})^j b_k.$$

Bounding s_k With some slight abuse of notation, we have

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

and thus:

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

$$= (\gamma P_{\pi_k})^m \Big(v_{k-1} - (T_{\pi_k})^\infty v_{k-1} \Big) = (\gamma P_{\pi_k})^m \sum_{j=0}^{\infty} \Big[(T_{\pi_k})^j v_{k-1} - (T_{\pi_k})^{j+1} v_{k-1} \Big]$$

$$= (\gamma P_{\pi_k})^m \Big(\sum_{j=0}^{\infty} \Big[(T_{\pi_k})^j v_{k-1} - (T_{\pi_k})^j T_{\pi_k} v_{k-1} \Big] = (\gamma P_{\pi_k})^m \Big(\sum_{j=0}^{\infty} (\gamma P_{\pi_k})^j \Big) (v_{k-1} - T_{\pi_k} v_{k-1})$$

$$= (\gamma P_{\pi_k})^m (I - \gamma P_{\pi_k})^{-1} (v_{k-1} - T_{\pi_k} v_{k-1}) = (\gamma P_{\pi_k})^m (I - \gamma P_{\pi_k})^{-1} b_k. \tag{4.31}$$

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

B Proof of Lemma 4.2

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

By induction, from Equations 4.25 and 4.26, we obtain

$$b_k \leq \sum_{i=1}^{k} \Gamma^{m(k-i)} x_i + \Gamma^{mk} b_0, \tag{4.32}$$

$$d_k \le \sum_{j=0}^{k-1} \Gamma^{k-1-j}\Big(y_j + \sum_{l=1}^{m-1} \Gamma^l b_j \Big) + \Gamma^k d_0. \tag{4.33}$$

in which we have used the notation introduced in Definition 4.1. In Equation 4.33, we also used the fact that from Equation 4.29, we may write $g_{k+1} = \sum_{j=1}^{m-1} \Gamma^j b_k$. Moreover, we may rewrite Equation 4.31 as

$$s_k = \Gamma^m \sum_{j=0}^{\infty} \Gamma^j b_{k-1} = \sum_{j=0}^{\infty} \Gamma^{m+j} b_{k-1}. \tag{4.34}$$

Bounding l_k From Equations 4.32 and 4.33, we may write

$$\begin{aligned} d_k &\le \sum_{j=0}^{k-1} \Gamma^{k-1-j} \left(y_j + \sum_{l=1}^{m-1} \Gamma^l \Big(\sum_{i=1}^{j} \Gamma^{m(j-i)} x_i + \Gamma^{mj} b_0 \Big) \right) + \Gamma^k d_0 \\ &= \sum_{i=1}^{k} \Gamma^{i-1} y_{k-i} + \sum_{j=0}^{k-1} \sum_{l=1}^{m-1} \sum_{i=1}^{j} \Gamma^{k-1-j+l+m(j-i)} x_i + z_k, \end{aligned} \tag{4.35}$$

where we used the following definition

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

The triple sum involved in Equation 4.35 may be written as

$$\begin{aligned} \sum_{j=0}^{k-1} \sum_{l=1}^{m-1} \sum_{i=1}^{j} \Gamma^{k-1-j+l+m(j-i)} x_i &= \sum_{i=1}^{k-1} \sum_{j=i}^{k-1} \sum_{l=1}^{m-1} \Gamma^{k-1+l+j(m-1)-mi} x_i = \sum_{i=1}^{k-1} \sum_{j=mi+k-i}^{mk-1} \Gamma^{j-mi} x_i \\ &= \sum_{i=1}^{k-1} \sum_{j=k-i}^{m(k-i)-1} \Gamma^j x_i = \sum_{i=1}^{k-1} \sum_{j=i}^{mi-1} \Gamma^j x_{k-i}. \end{aligned} \tag{4.36}$$

Using Equation 4.36, we may write Equation 4.35 as

$$d_k \le \sum_{i=1}^{k} \Gamma^{i-1} y_{k-i} + \sum_{i=1}^{k-1} \sum_{j=i}^{mi-1} \Gamma^j x_{k-i} + z_k. \tag{4.37}$$

Similarly, from Equations 4.34 and 4.32, we have

$$\begin{aligned} s_k &\le \sum_{j=0}^{\infty} \Gamma^{m+j} \Big(\sum_{i=1}^{k-1} \Gamma^{m(k-1-i)} x_i + \Gamma^{m(k-1)} b_0 \Big) \\ &= \sum_{j=0}^{\infty} \Big(\sum_{i=1}^{k-1} \Gamma^{m+j+m(k-1-i)} x_i + \Gamma^{m+j+m(k-1)} b_0 \Big) \\ &= \sum_{i=1}^{k-1} \sum_{j=0}^{\infty} \Gamma^{j+m(k-i)} x_i + \sum_{j=0}^{\infty} \Gamma^{j+mk} b_0 = \sum_{i=1}^{k-1} \sum_{j=0}^{\infty} \Gamma^{j+mi} x_{k-i} + \sum_{j=mk}^{\infty} \Gamma^j b_0 \\ &= \sum_{i=1}^{k-1} \sum_{j=mi}^{\infty} \Gamma^j x_{k-i} + z'_k, \end{aligned} \tag{4.38}$$

where we used the following definition

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

Finally, using the bounds in Equations 4.37 and 4.38, we obtain the following bound on the loss

$$\begin{aligned} l_k \le d_k + s_k &\le \sum_{i=1}^{k} \Gamma^{i-1} y_{k-i} + \sum_{i=1}^{k-1}\Big(\sum_{j=i}^{mi-1}\Gamma^j + \sum_{j=mi}^{\infty}\Gamma^j\Big)x_{k-i} + z_k + z'_k \\ &= \sum_{i=1}^{k} \Gamma^{i-1} y_{k-i} + \sum_{i=1}^{k-1}\sum_{j=i}^{\infty}\Gamma^j x_{k-i} + \eta_k, \end{aligned} \tag{4.39}$$

where we used the following definition

$$\eta_k \triangleq z_k + z'_k = \sum_{j=k}^{\infty}\Gamma^j b_0 + \Gamma^k d_0. \tag{4.40}$$

Note that we have the following relation between b_0 and d_0

$$b_0 = v_0 - T_{\pi_1}v_0 = v_0 - v^* + T_{\pi^*}v^* - T_{\pi^*}v_0 + T_{\pi^*}v_0 - T_{\pi_1}v_0 \le (I-\gamma P_{\pi^*})(-d_0) + \epsilon'_1, \tag{4.41}$$

In Equation 4.41, we used the fact that $v^* = T_{\pi^*}v^*$, $\epsilon_0 = 0$, and $T_{\pi^*}v_0 - T_{\pi_1}v_0 \le \epsilon'_1$ (this is because the policy π_1 is ϵ'_1-greedy w.r.t. v_0). As a result, we may write $|\eta_k|$ either as

$$\begin{aligned} |\eta_k| &\le \sum_{j=k}^{\infty}\Gamma^j\Big[(I-\gamma P_{\pi^*})|d_0| + |\epsilon'_1|\Big] + \Gamma^k|d_0| \\ &\le \sum_{j=k}^{\infty}\Gamma^j\Big[(I+\Gamma^1)|d_0| + |\epsilon'_1|\Big] + \Gamma^k|d_0| = 2\sum_{j=k}^{\infty}\Gamma^j|d_0| + \sum_{j=k}^{\infty}\Gamma^j|\epsilon'_1|, \end{aligned} \tag{4.42}$$

or using the fact that from Equation 4.41, we have $d_0 \le (I-\gamma P_{\pi^*})^{-1}(-b_0 + \epsilon'_1)$, as

$$\begin{aligned} |\eta_k| &\le \sum_{j=k}^{\infty}\Gamma^j|b_0| + \Gamma^k\sum_{j=0}^{\infty}(\gamma P_{\pi^*})^j\Big(|b_0| + |\epsilon'_1|\Big) \\ &= \sum_{j=k}^{\infty}\Gamma^j|b_0| + \Gamma^k\sum_{j=0}^{\infty}\Gamma^j\Big(|b_0| + |\epsilon'_1|\Big) = 2\sum_{j=k}^{\infty}\Gamma^j|b_0| + \sum_{j=k}^{\infty}\Gamma^j|\epsilon'_1|. \end{aligned} \tag{4.43}$$

Now, using the definitions of x_k and y_k in Equations 4.24 and 4.28, the bound on $|\eta_k|$ in Equation 4.42 or 4.43, and the fact that $\epsilon_0 = 0$, we obtain

$$\begin{aligned}
|l_k| &\le \sum_{i=1}^{k} \Gamma^{i-1}\Big[\Gamma^1|\epsilon_{k-i}| + |\epsilon'_{k-i+1}|\Big] + \sum_{i=1}^{k-1}\sum_{j=i}^{\infty}\Gamma^j\Big[(I+\Gamma^1)|\epsilon_{k-i}| + |\epsilon'_{k-i+1}|\Big] + |\eta_k| \\
&= \sum_{i=1}^{k-1}\left(\Gamma^i + \sum_{j=i}^{\infty}(\Gamma^j + \Gamma^{j+1})\right)|\epsilon_{k-i}| + \Gamma^k|\epsilon_0| \qquad (4.44)\\
&\quad + \sum_{i=1}^{k-1}\left(\Gamma^{i-1} + \sum_{j=i}^{\infty}\Gamma^j\right)|\epsilon'_{k-i+1}| + \Gamma^{k-1}|\epsilon'_1| + \sum_{j=k}^{\infty}\Gamma^j|\epsilon'_1| + h(k) \\
&= 2\sum_{i=1}^{k-1}\sum_{j=i}^{\infty}\Gamma^j|\epsilon_{k-i}| + \sum_{i=1}^{k-1}\sum_{j=i-1}^{\infty}\Gamma^j|\epsilon'_{k-i+1}| + \sum_{j=k-1}^{\infty}\Gamma^j|\epsilon'_1| + h(k) \\
&= 2\sum_{i=1}^{k-1}\sum_{j=i}^{\infty}\Gamma^j|\epsilon_{k-i}| + \sum_{i=0}^{k-1}\sum_{j=i}^{\infty}\Gamma^j|\epsilon'_{k-i}| + h(k), \qquad (4.45)
\end{aligned}$$

where we used the following definition

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

We end this proof by adapting the error propagation to CBMPI. As expressed by Equations 4.17 and 4.18 in Section 3, an analysis of CBMPI can be deduced from that we have just done by replacing v_k with the auxiliary variable $w_k = (T_{\pi_k})^m v_{k-1}$ and ϵ_k with $(\gamma P_{\pi_k})^m \epsilon_{k-1} = \Gamma^m\epsilon_{k-1}$. Therefore, using the fact that $\epsilon_0 = 0$, we can rewrite the bound of Equation 4.45 for CBMPI as follows:

$$\begin{aligned}
l_k &\le 2\sum_{i=1}^{k-1}\sum_{j=i}^{\infty}\Gamma^{j+m}|\epsilon_{k-i-1}| + \sum_{i=0}^{k-1}\sum_{j=i}^{\infty}\Gamma^j|\epsilon'_{k-i}| + h(k) \\
&= 2\sum_{i=1}^{k-2}\sum_{j=m+i}^{\infty}\Gamma^j|\epsilon_{k-i-1}| + \sum_{i=0}^{k-1}\sum_{j=i}^{\infty}\Gamma^j|\epsilon'_{k-i}| + h(k). \qquad (4.46)
\end{aligned}$$

C Proof of Lemma 4.3

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

$$\rho\Gamma^t|z| = \left\|\Gamma^t|z|\right\|_{1,\rho} \le \gamma^t c_q(t)\|z\|_{q',\mu} = \gamma^t c_q(t)\left(\mu|z|^{q'}\right)^{\frac{1}{q'}}. \qquad (4.47)$$

We define

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

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

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

where **(a)** results from Jensen's inequality, **(b)** from Equation 4.47, and **(c)** from the definition of $\mathcal{C}_q(l)$. Now, by setting $\xi_l = \left(\mathcal{C}_q(l)\right)^{1/p} \sup_{i\in\mathcal{I}_l} \|g_i\|_{pq',\mu}$, we obtain

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

where the last step follows from the definition of K.

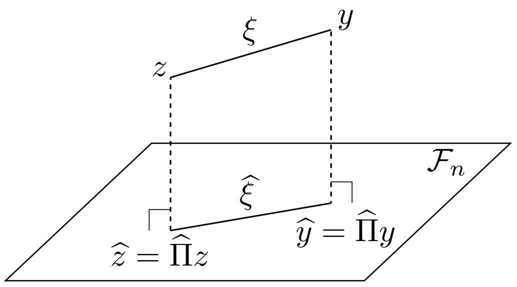
D Proof of Theorem 4.1 & Another Bounds on the Loss

Proof. We only detail the proof for AMPI (the proof being similar for CBMPI). We define $\mathcal{I} = \{1, 2, \cdots, 2k\}$, the partition $\mathcal{I} = \{\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3\}$ as $\mathcal{I}_1 = \{1, \dots, k-1\}$, $\mathcal{I}_2 = \{k, \dots, 2k-1\}$, and $\mathcal{I}_3 = \{2k\}$, and for each $i \in \mathcal{I}$

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

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

Note that here we have divided the terms in the point-wise bound of Lemma 4.2 into three groups: the *evaluation error* terms $\{\epsilon_j\}_{j=1}^{k-1}$, the *greedy step error* terms $\{\epsilon'_j\}_{j=1}^k$,



ξ
y
z
$\mathcal{F}_n$
$\widehat{\xi}$
$\widehat{y} = \widehat{\Pi} y$
$\widehat{z} = \widehat{\Pi} z$

Figure 4.11: The vectors used in the proof.

and finally the residual term $h(k)$. With the above definitions and the fact that the loss l_k is non-negative, Lemma 4.2 may be rewritten as

$$|l_k| \leq \sum_{l=1}^{3} \sum_{i \in \mathcal{I}_l} \sum_{j \in \mathcal{J}_i} \Gamma^j |g_i|.$$

The result follows by applying Lemma 4.3 and noticing that $\sum_{i=i_0}^{k-1} \sum_{j=i}^{\infty} \gamma^j = \frac{\gamma^{i_0} - \gamma^k}{(1-\gamma)^2}$. □

Here in order to show the flexibility of Lemma 4.3, we group the terms differently and derive an alternative ℓ_p-bound for the loss of AMPI and CBMPI. In analogy with the results of Farahmand et al. (2010), this new bound shows that the last iterations have the highest influence on the loss (the influence exponentially decreases towards the initial iterations).

Theorem 4.3. *With the notations of Theorem 4.1, after k iterations, the loss of AMPI satisfies*

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

while the loss of CBMPI satisfies

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

Proof. Again, we only detail the proof for AMPI (the proof being similar for CBMPI). We define $\mathcal{I}$, (g_i) and $(\mathcal{J}_i)$ as in the proof of Theorem 4.1. We then make as many groups as terms, i.e., for each $n \in \{1, 2, \ldots, 2k-1\}$, we define $\mathcal{I}_n = \{n\}$. The result follows by application of Lemma 4.3. □

E Proof of Lemma 4.4

Let us define two N-dimensional vectors $z = \left(\left[(T_{\pi_k})^m v_{k-1}\right](s^{(1)}), \ldots, \left[(T_{\pi_k})^m v_{k-1}\right](s^{(N)})\right)^\top$ and $y = \left(\widehat{v}_k(s^{(1)}), \ldots, \widehat{v}_k(s^{(N)})\right)^\top$ and their orthogonal projections onto the vector space $\mathcal{F}_N$ as $\widehat{z} = \widehat{\Pi} z$ and

$\widehat{y} = \widehat{\Pi} y = \Big(\widetilde{v}_k(s^{(1)}), \dots, \widetilde{v}_k(s^{(N)})\Big)^\top$, where $\widetilde{v}_k$ is the result of linear regression and its truncation (by $V_{\max}$) is v_k, i.e., $v_k = \mathbb{T}(\widetilde{v}_k)$ (see Figure 4.11). What we are interested is to find a bound on the regression error $\|z - \widehat{y}\|$ (the difference between the target function z and the result of the regression $\widehat{y}$). We may decompose this error as

$$\|z - \widehat{y}\|_N \leq \|\widehat{z} - \widehat{y}\|_N + \|z - \widehat{z}\|_N = \|\widehat{\xi}\|_N + \|z - \widehat{z}\|_N, \tag{4.48}$$

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

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

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

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

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

$$\|\widehat{z} - \widehat{y}\|_N = \|\widehat{\xi}\|_N \leq 4V_{\max} \sqrt{\frac{2}{N} \log\left(\frac{3(9e^2N)^{d+1}}{\delta'}\right)}. \tag{4.49}$$

From Equations 4.48 and 4.49, we have

$$\|(T_{\pi_k})^m v_{k-1} - \widetilde{v}_k\|_{\widehat{\mu}} \leq \|(T_{\pi_k})^m v_{k-1} - \widehat{\Pi}(T_{\pi_k})^m v_{k-1}\|_{\widehat{\mu}} + 4V_{\max} \sqrt{\frac{2}{N} \log\left(\frac{3(9e^2N)^{d+1}}{\delta'}\right)}, \tag{4.50}$$

where $\widehat{\mu}$ is the empirical norm induced from the N i.i.d. samples from μ.

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

$$\begin{aligned}\|f_{\widehat{\alpha}_*} - (T_{\pi_k})^m v_{k-1}\|_{\widehat{\mu}} \leq \|f_{\alpha_*} - (T_{\pi_k})^m v_{k-1}\|_{\widehat{\mu}} &\leq 2\|f_{\alpha_*} - (T_{\pi_k})^m v_{k-1}\|_\mu \\ &+ 12\Big(V_{\max} + \|\alpha_*\|_2 \sup_x \|\phi(x)\|_2\Big) \sqrt{\frac{2}{N} \log \frac{3}{\delta'}}\ ,\end{aligned} \tag{4.51}$$

with probability at least $1 - \delta'$, where the second inequality is the application of a variation of Theorem 11.2 in the book by Györfi et al., (2002) with $\|f_{\alpha_*} - (T_{\pi_k})^m v_{k-1}\|_\infty \leq$

$V_{\max} + \|\alpha^*\|_2 \, \sup_x \|\phi(x)\|_2$. Similarly, we can write the left-hand-side of Equation 4.50 as

$$2\|(T_{\pi_k})^m v_{k-1} - \widetilde{v}_k\|_{\widehat{\mu}} \geq 2\|(T_{\pi_k})^m v_{k-1} - \mathbb{T}(\widetilde{v}_k)\|_{\widehat{\mu}} \tag{4.52}$$

$$\geq \|(T_{\pi_k})^m v_{k-1} - \mathbb{T}(\widetilde{v}_k)\|_{\mu} - 24V_{\max}\sqrt{\frac{2}{N}\Lambda(N,d,\delta')}, \tag{4.53}$$

with probability at least $1-\delta'$, where $\Lambda(N,d,\delta') = 2(d+1)\log N + \log\frac{e}{\delta'} + \log\Big(9(12e)^{2(d+1)}\Big)$. Putting together Equations 4.50, 4.51, and 4.52 and using the fact that $\mathbb{T}(\widetilde{v}_k) = v_k$, we obtain

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

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

F Proof of Lemma 4.5

Proof. For each $s \in \mathcal{S}$, we may write

$$\begin{aligned}|\epsilon'_k(s)| &\overset{(a)}{=} \max_{a\in\mathcal{A}} \Big(T_a v_{k-1}\Big)(s) - \Big(T_{\pi_k} v_{k-1}\Big)(s) \\ &\overset{(b)}{\leq} \max_{a\in\mathcal{A}} \Big(T_a v_{k-1}\Big)(s) - \Big(\widehat{T}_{\pi_k} v_{k-1}\Big)(s) + \gamma V_{\max}\sqrt{\frac{2\log(1/\delta')}{M}} && \text{w.p. } 1-\delta' \\ &\overset{(c)}{=} \max_{a\in\mathcal{A}} \Big(T_a v_{k-1}\Big)(s) - \max_{a'\in\mathcal{A}} \Big(\widehat{T}_{a'} v_{k-1}\Big)(s) + \gamma V_{\max}\sqrt{\frac{2\log(1/\delta')}{M}} && \text{w.p. } 1-\delta' \\ &\overset{(d)}{\leq} \max_{a\in\mathcal{A}} \Big[\Big(T_a v_{k-1}\Big)(s) - \Big(\widehat{T}_a v_{k-1}\Big)(s)\Big] + \gamma V_{\max}\sqrt{\frac{2\log(1/\delta')}{M}} && \text{w.p. } 1-\delta' \\ &\overset{(e)}{\leq} \gamma V_{\max}\sqrt{\frac{2\log(|\mathcal{A}|/\delta')}{M}} + \gamma V_{\max}\sqrt{\frac{2\log(1/\delta')}{M}} && \text{w.p. } 1-2\delta' \\ &\leq 2\gamma V_{\max}\sqrt{\frac{2\log(|\mathcal{A}|/\delta')}{M}} \;. && \text{w.p. } 1-2\delta'\end{aligned}$$

(a) This is from the definition of the greedy step error ϵ'_k.

(b) Here we use a Chernoff-Hoeffding bound and replace

$$\Big(T_{\pi_k} v_{k-1}\Big)(s) = r\Big(s, \pi_k(s)\Big) + \gamma \int P\Big(ds'|s, \pi_k(s)\Big) v_{k-1}(s')$$

with its empirical version

$$\big(\widehat{T}_{\pi_k} v_{k-1}\big)(s) = r\big(s, \pi_k(s)\big) + \frac{\gamma}{M}\sum_{j=1}^{M} v(s'^{(j)}), \qquad \{s'^{(j)}\}_{j=1}^{M} \overset{\text{i.i.d.}}{\sim} P\big(\cdot\,|s, \pi_k(s)\big).$$

We have

$$\begin{aligned}\big(\widehat{T}_{\pi_k} v_{k-1}\big)(s) - \big(T_{\pi_k} v_{k-1}\big)(s) &= \gamma\Big(\frac{1}{M}\sum_{j=1}^{M} v(s'^{(j)}) - \mathbb{E}_{s'\sim P(\cdot|s,\pi_k(s))}\big[v_{k-1}(s')\big]\Big) \\ &\leq \gamma V_{\max}\sqrt{\frac{2\log(1/\delta)}{M}} \qquad \text{w.p. } 1-\delta'.\end{aligned}$$

(c) This step is from the definition of π_k in the AMPI-V algorithm (Equation 4.1).

(d) This step is algebra, replacing two maximums with one.
(e) Finally, this step is another Chernoff-Hoeffding bound similar to Step (b) plus a union bound over actions

$$\max_{a\in\mathcal{A}}\Big[\big(T_a v_{k-1}\big)(s) - \big(\widehat{T}_a v_{k-1}\big)(s)\Big] \leq \gamma V_{\max}\sqrt{\frac{2\log(|\mathcal{A}|/\delta')}{M}} \qquad \text{w.p. } 1-\delta'.$$

□

G Proof of Lemma 4.6

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

Lemma 4.7. *Let Π be a policy space with finite VC-dimension $h = VC(\Pi) < \infty$ and N' be the number of states in the rollout set $\mathcal{D}'_{k-1}$ drawn i.i.d. from the state distribution μ. Then we have*

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

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

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

Lemma 4.8. *Let Π be a policy space with finite VC-dimension $h = VC(\Pi) < \infty$ and $s^{(1)}, \ldots, s^{(N')}$ be an arbitrary sequence of states. At each state we simulate M independent rollouts of the form , then we have*

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

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

Proof. The proof is similar to the one for Lemma 4.7. □

Proof. **(Lemma 4.6)** Let $a^*(\cdot) \in \operatorname{argmax}_{a\in\mathcal{A}} Q_{k-1}(\cdot, a)$ be the greedy action. To simplify the notation, we remove the dependency of a^* on states and use a^* instead of $a^*(s^{(i)})$ in the following. We prove the following series of inequalities:

$$
\begin{aligned}
&\mathcal{L}^{\Pi}_{k-1}(\mu;\pi_k) \\
&\quad \overset{(a)}{\leq} \mathcal{L}^{\Pi}_{k-1}(\widehat{\mu};\pi_k) + e'_1(N',\delta) \qquad \text{w.p. } 1-\delta' \\
&\quad = \frac{1}{N'}\sum_{i=1}^{N'}\Big[Q_{k-1}(s^{(i)},a^*) - Q_{k-1}\big(s^{(i)},\pi_k(s^{(i)})\big)\Big] + e'_1(N',\delta) \\
&\quad \overset{(b)}{\leq} \frac{1}{N'}\sum_{i=1}^{N'}\Big[Q_{k-1}(s^{(i)},a^*) - \widehat{Q}_{k-1}\big(s^{(i)},\pi_k(s^{(i)})\big)\Big] \\
&\qquad + e'_1(N',\delta) + e'_2(N',M,\delta) \qquad \text{w.p. } 1-2\delta' \\
&\quad \overset{(c)}{\leq} \frac{1}{N'}\sum_{i=1}^{N'}\Big[Q_{k-1}(s^{(i)},a^*) - \widehat{Q}_{k-1}\big(s^{(i)},\pi^*(s^{(i)})\big)\Big] + e'_1(N',\delta) + e'_2(N',M,\delta) \\
&\quad \leq \frac{1}{N'}\sum_{i=1}^{N'}\Big[Q_{k-1}(s^{(i)},a^*) - Q_{k-1}\big(s^{(i)},\pi^*(s^{(i)})\big)\Big] \\
&\qquad + e'_1(N',\delta) + 2e'_2(N',M,\delta) \qquad \text{w.p. } 1-3\delta' \\
&\quad = \mathcal{L}^{\Pi}_{k-1}(\widehat{\mu};\pi^*) + e'_1(N',\delta) + 2e'_2(N',M,\delta) \\
&\quad \leq \mathcal{L}^{\Pi}_{k-1}(\mu;\pi^*) + 2\Big(e'_1(N',\delta) + e'_2(N',M,\delta)\Big) \qquad \text{w.p. } 1-4\delta' \\
&\quad = \inf_{\pi\in\Pi}\mathcal{L}^{\Pi}_{k-1}(\mu;\pi) + 2\Big(e'_1(N',\delta) + e'_2(N',M,\delta)\Big).
\end{aligned}
$$

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

(a) This follows from Lemma 4.7.
(b) Here we introduce the estimated action-value function $\widehat{Q}_{k-1}$ by bounding

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

using Lemma 4.8.
(c) From the definition of π_k in CBMPI, we have

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

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

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

□

Chapter 5
Multi-Bandit Best Arm Identification

In this chapter,[1] we study the problem of identifying the best arm in each of the bandits in a multi-bandit multi-armed problem, in both *fixed budget* and *fixed confidence* settings. We first propose two algorithms, called Gap-based Exploration (GapE) and Unified Gap-based Exploration (UGapE), that are both based on the idea of selecting the arm whose mean is close to the mean of the best arm in the same bandit (i.e., small gap). We then introduce an algorithm, called GapE-V (respectively UGapE-V), which takes into account the variance of the arms in addition to their gap. We prove an upper-bound on the probability of error for all these algorithms. Since in the fixed budget setting, GapE and GapE-V need to tune an exploration parameter that depends on the complexity of the problem and is often unknown in advance, we also introduce variations of these algorithms that estimate this complexity online. Finally, we evaluate the performance of these algorithms and compare them to other allocation strategies, in a number of synthetic problems, using real-word clinical data, and finally in the rollout allocation step of the classification-based policy iteration (CBPI) algorithms (see the discussion in Section 1.3.2 of Chapter 2).

Contents

[1]The chapter is an extended version of our two NIPS publications (Gabillon et al., 2011a, 2012).

1 Introduction and Motivating Examples

We begin by giving several examples to motivate the study of the multi-bandit multi-armed setting. Consider a clinical problem with M subpopulations, in which one should decide between K_p options for treating subjects from each subpopulation p. A subpopulation may correspond to patients with a particular gene biomarker (or other risk categories) and the treatment options are the available treatments for a disease. The main objective here is to construct a rule, which recommends the best treatment for each of the subpopulations. These rules are usually constructed using data from clinical trials that are generally costly to run. Therefore, it is important to distribute the trial resources wisely so that the devised rule yields a good performance. Since it may take significantly more resources to find the best treatment for one subpopulation than for the others, the common strategy of enrolling patients as they arrive may not yield an overall good performance. Moreover, applying treatment options uniformly at random in a subpopulation could not only waste trial resources, but also it might run the risk of finding a bad treatment for that subpopulation. This problem can be formulated as the *best arm identification* (Audibert et al., 2010) over M multi-armed bandits, which in turn can be seen as the problem of *pure exploration* (Bubeck et al., 2009) over multiple bandits. In this formulation, each subpopulation is considered as a multi-armed bandit, each treatment as an arm, trying a medication on a patient as a pull, and we are asked to recommend an arm for each bandit after a given number of pulls (budget). The evaluation can be based on **1)** the average over the bandits of the reward of the recommended arms, or **2)** the average probability of error (not selecting the best arm), or **3)** the maximum probability of error. Note that this setting is different from the standard multi-armed bandit problem in which the goal is to maximize the cumulative sum of rewards (see e.g., Robbins 1952, Auer et al. 2002).

The second example is the popular problem of online advertisement, where a company uses a testing phase before deploying its advertisement system. This problem can also be formulated as above, where each bandit is a subpopulation of Internet users (e.g., young, old, single, married), each arm is a category of advertisements, and each pull is to show an advertisement to a user. Here the goal is to actively learn a rule, which recommends the best (the one with the highest chance to be clicked on) category of advertisements for each of the subpopulations.

Another example is the following brain-computer interface (BCI) problem. A computer has to guess a letter chosen by a user. The computer arranges the letters in a matrix displayed to the user as shown in Figure 5.1. At each time step, the computer chooses either a row or a column and asks the user if the chosen letter belongs to it. The answer is obtained by recording noisy brain activity signals. This problem can be formalized as a two-bandit best arm identification problem where the bandits are "rows" and "columns". In this problem, the right measure of performance is exactly the maximum probability of error, since doing a mistake in either row or column would lead to choose the wrong letter.

Finally, the problem that motivated us to study multi-bandit best arm identifi-

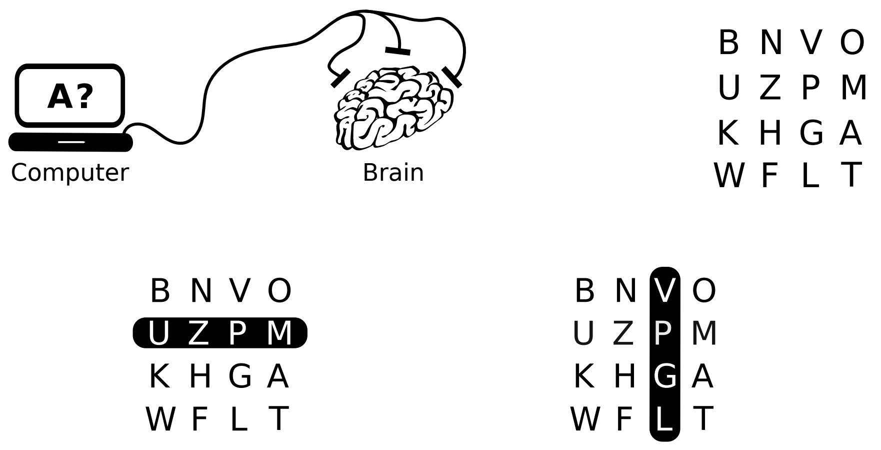


Figure 5.1: The BCI problem in which the goal is to design a strategy of repetitively flashing rows and columns of a grid of letters in order to identify the letter chosen by the user.

cation, namely the rollout allocation problem in Classification-based Policy Iteration (CBPI) algorithms (see Section 1.3.2 of Chapter 2 for a detailed description). In this problem, the goal is to identify the best action (arm) in each of the states (bandit) in the rollout set. We will discuss this application in more details in Section 5.1.

The problem of *best arm(s) identification* (Even-Dar et al., 2006, Bubeck et al., 2009, Audibert et al., 2010) in stochastic multi-armed bandit has recently received much attention. In this problem, during an *exploration* phase, a forecaster repeatedly selects an arm and observes a sample drawn from the reward distribution of this arm, and when the exploration phase ends, it is asked to return the arm(s) with the highest mean value(s). Unlike the standard (cumulative regret) setting, where the goal is to maximize the cumulative sum of rewards obtained by the forecaster (see e.g., Robbins 1952, Auer et al. 2002), in this setting the forecaster is evaluated on the quality of the arm(s) it returns at the end of the exploration phase. As discussed in Section 2.3.1 of Chapter 2, the problem of best arm(s) identification has been studied in two distinct settings in the literature. In the *fixed budget* setting (see e.g., Bubeck et al. 2009, Audibert et al. 2010), the number of rounds of the exploration phase is fixed and is known by the forecaster, and the objective is to maximize the probability of returning the best arm(s). In the *fixed confidence* setting (see e.g., Maron and Moore 1993, Even-Dar et al. 2006), the forecaster aims to minimize the number of rounds needed to achieve a fixed confidence about the quality of the returned arm(s).

We first became interested in studying the multi-bandit best arm identification problem in the fixed budget setting. Audibert et al. (2010) proposed two algorithms for the *fixed budget* setting: **1)** a highly exploring strategy based on upper confidence bounds, called *upper confidence bound – exploration* (UCB-E), in which the optimal value of its parameter depends on some measure of the complexity of the problem that is unknown in advance, and **2)** a parameter-free method based on progressively rejecting the arms that seem to be sub-optimal, called *Successive Rejects* (SR). They showed that both algorithms are nearly optimal since their probability of returning the wrong arm decreases exponentially. However, UCB-E and SR are designed for the single-bandit problem, and we argue that their trivial extensions are not good enough (sub-optimal) for the multi-bandit scenario. A naive application of UCB-E (or SR) in the multi-bandit setting would pull more the arms with the highest estimated mean over all of the arms of all the bandits. Therefore, the resulting algorithm would only identify the best arm in all the bandits and not the best arm in each bandit. Another approach is to extend the Hoeffding Races algorithm, first introduced by Maron and Moore (1993) in the fixed confidence setting, to the multi-bandit case with fixed budget. Nevertheless, as shown by Audibert et al. (2010) and more recently by Kaufmann and Kalyanakrishnan (2013), Hoeffding Races belong to the family of the *uniform sampling* strategies (strategies that sample uniformly over the arms that have not been discarded yet) that are not efficient, neither in the fixed budget nor in the fixed confidence setting, compared to the *adaptive sampling* strategies (that typically sample the arms according to an index measuring how "important" is to sample each arm). Therefore, with the objective to design efficient index-based strategies, we propose to extend the UCB-E algorithm to the multi-bandit scenario under both fixed budget and fixed confidence settings. Moreover, we address the case where the m best arms (with $m \geq 1$) must be identified (a problem also studied in the fixed confidence setting by Kalyanakrishnan and Stone 2010) and the case where it is sufficient to identify an arm whose mean is ϵ-close to the one of the best arm. Note that prior to our work, the multi-bandit setting was only studied by Deng et al. (2011), who proposed an active learning algorithm based on an ϵ-greedy heuristic, but did not provide any theoretical analysis for the algorithm and only empirically evaluated its performance. After our work, Wang et al. (2013) also studied the multi-bandit setting and the problem of m-best arm identification, and proposed a new version of Successive Rejects, called SAR (with accept and reject), that is able to return the set of the m-best arms with high probability.

We propose two algorithms, called Gap-based Exploration (GapE) and Unified GapE (UGapE). The allocation strategy implemented by (U)GapE focuses on the gap of the arms, i.e., the difference between the mean of the arm and the mean of the best arm in that bandit. We also propose the (U)GapE-variance algorithm, with the abbreviation (U)GapE-V, that extends this approach by taking into account the variance of the arms. For all the algorithms, in the fixed budget setting, we prove an upper-bound on the probability of error that decreases exponentially with the budget. This bound is the first theoretical guarantee proved in the multi-bandit setting. Since in the fixed budget, both (U)GapE and (U)GapE-V need to tune an exploration parameter that depends on the complexity of the problem, which is rarely known in advance, we also introduce their adaptive version that learns this complexity on-line. Finally, we evaluate the performance of these algorithms and compare them with *Uniform* and *Uniform+UCB-E* strategies on a number of synthetic problems. Our empirical results indicate that **1)** (U)GapE and (U)GapE-V have a better performance than *Uniform* and *Uniform+UCB-E*, and **2)** the adaptive version of these algorithms match the performance of their non-adaptive counterparts. Finally we apply our strategies to some of the problems that motivated our interest. We show that our strategies

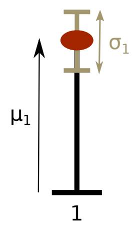


Figure 5.2: Graphical notations: this figure shows how we illustrate the mean and variance of an arm indexed as *arm* 1.

outperform the heuristic proposed by Deng et al. (2011) in a real world clinical problem. The application to the CBPI algorithms' rollout allocation problem is also discussed.

2 Problem Formulation

In this section, we first introduce the notation used throughout the chapter. We also formally introduce the multi-bandit (ϵ, m)-best arm identification problem.

Let M be the number of bandits. For simplicity and without loss of generality, let K be the number of arms for each bandit. We use indices p and q for the bandits and k, i, and j for the arms. Let $A = \{1, \dots, K\}$ be the set of indices of the arms in each bandit such that each arm $k \in A$ of bandit p is characterized by a distribution ν_{pk} bounded in $[0, b]$ with mean μ_{pk} and variance σ^2_{pk}. In Figure 5.2, we show how we graphically illustrate the mean and variance of an arm. This will help us later to graphically represent the complexities of the bandit problems used as illustrative examples or in the experiments.

We define the m-max and m-argmax operators as[2]

$$\mu_{p(m)} = \overset{m}{\max_{k \in A}}\ \mu_{pk} \qquad \text{and} \qquad (m)_p = \arg \overset{m}{\max_{k \in A}}\ \mu_{pk}\ ,$$

where $(m)_p$ denotes the index of the m-th best arm in A for bandit p and $\mu_{p(m)}$ is its corresponding mean so that $\mu_{p(1)} \geq \mu_{p(2)} \geq \dots \geq \mu_{p(K)}$. We denote by $S^{pm} \subset A$ any subset of m arms of bandit p (i.e., $|S^{pm}| = m < K$) and by $S^{pm,*}$ the subset of the m best arms of bandit p (i.e., $k \in S^{pm,*}$ iif $\mu_{pk} \geq \mu_{p(m)}$). Without loss of generality, we assume that there exists, for any bandit p, a unique set $S^{pm,*}$. In the following we drop the superscript m and use $S^p = S^{pm}$ and $S^{p,*} = S^{pm,*}$ whenever m is clear from the context. With a slight abuse of notation we further extend the m-max operator to an operator returning a set of arms, such that

$$\{\mu_{p(1)}, \dots, \mu_{p(m)}\} = \overset{1..m}{\max_{k \in A}}\ \mu_{pk} \qquad \text{and} \qquad S^{p,*} = \arg \overset{1..m}{\max_{k \in A}}\ \mu_{pk}\ .$$

Finally in each bandit p, we define the gap for the k-th arm as $\Delta_{pk} = |\overset{m}{\max_{j \neq k}} \mu_{pj} - \mu_{pk}|$.

[2]Ties are broken in an arbitrary but consistent manner.

Given an accuracy ϵ and a number of arms m, we say that an arm k in bandit p is (ϵ,m)-optimal if $\mu_{pk} \geq \mu_{p(m)} - \epsilon$. Thus, we define the multi-bandit (ϵ,m)-best arm identification problem as the problem of finding a set S of m (ϵ,m)-optimal arms in each of the M bandits. Once again, for simplicity and without loss of generality, the parameter m is the same for all the bandits. The clinical trial problem described in Section 1 is an $(\epsilon = 0, m = 1)$-multi-bandit best arm identification problem, which can be formalized as a game between a stochastic bandit environment and a forecaster. The distributions $\{\nu_{pk}\}$ are unknown to the forecaster. At each round t, the forecaster pulls an arm $I(t) \in A$ in bandit $q(t)$ and observes an independent sample drawn from the distribution $\nu_{q(t),I(t)}$ independent from the past. The forecaster estimates the expected value of each arm by computing the average of the samples observed over time. Let $T_{pk}(t)$ be the number of times that arm k of bandit p has been pulled by the end of round t, then the mean of this arm is estimated as $\widehat{\mu}_{pk}(t) = \frac{1}{T_{pk}(t)} \sum_{s=1}^{T_{pk}(t)} X_{pk}(s)$, where $X_{pk}(s)$ is the s-th sample observed from ν_{pk}. Given the previous definitions, we define the estimated gaps as $\widehat{\Delta}_{pk}(t) = |\max^m_{j\neq k} \widehat{\mu}_{pj}(t) - \widehat{\mu}_{pk}(t)|$. For any arm $k \in A$ in bandit p, we define the notion of *arm simple regret* as

$$r_{pk} = \mu_{p(m)} - \mu_{pk}, \tag{5.1}$$

and for any set $S \subset A$ of m arms in bandit p, we define the *simple regret* as

$$r_S = \max_{k\in S} r_{pk} = \mu_{p(m)} - \min_{k\in S} \mu_{pk}. \tag{5.2}$$

We denote by $\Omega_p(t) \subset A$ the set of m arms of bandit p returned by the forecaster at the end of the exploration phase (when the algorithm stops after t rounds), and by $r_{\Omega_p(t)}$ its corresponding simple regret. Returning m (ϵ,m)-optimal arms is then equivalent to having $r_{\Omega_p(t)}$ smaller than ϵ. Finally, we can define the total regret incurred over all the bandits as,

$$r(t) = \frac{1}{M} \sum_{p=1}^{M} r_{\Omega_p(t)}.$$

As discussed in the introduction, other performance measures can be defined for this problem. In some applications, returning the wrong arm is considered as an error independently from its regret, and thus, the objective is to minimize the average probability of error

$$e(t) = \frac{1}{M} \sum_{p=1}^{M} e_p(t) = \frac{1}{M} \sum_{p=1}^{M} \mathbb{P}\Big(r_{\Omega_p(t)} \geq \epsilon\Big).$$

Finally, in problems similar to the brain computer interface, a reasonable objective is to return the correct letter, which requires the identification of both the correct column and the correct row, and not just to have a small average probability of error (averaged other row and column in the BCI case). In this case, the global performance of the forecaster can be measured as

$$\ell(t) = \max_p \ell_p(t) = \max_p \mathbb{P}\Big(r_{\Omega_p(t)} \geq \epsilon\Big).$$

```
Parameters: number of rounds $n$, exploration parameter $a$, maximum range $b$
Initialize: $T_{pk}(0) = 0$, $\widehat{\Delta}_{pk}(0) = 0$ for all bandit-arm pairs $(p, k)$
for $t = 1, 2, \dots, n$ do
    Compute $B_{pk}(t) = -\widehat{\Delta}_{pk}(t-1) + \beta_{pk}(t-1)$ for all bandit-arm pairs $(p, k)$
    Draw $(q(t), I(t)) \in \arg\max_{p,k} B_{pk}(t)$
    Observe $X_{q(t),I(t)}(T_{q(t),I(t)}(t-1)+1) \sim \nu_{q(t),I(t)}$
    Update $T_{q(t),I(t)}(t) = T_{q(t),I(t)}(t-1)+1$ and $\widehat{\Delta}_{q(t)k}(t)$ $\forall k$ of the selected bandit $q(t)$
end for
Return $J_p(n) \in \mathrm{argmax}_{k\in\{1,\dots,K\}} \widehat{\mu}_{pk}(n)$, $\forall p \in \{1 \dots M\}$
```

Figure 5.3: The pseudo-code of the gap-based Exploration (GapE) algorithm.

It is interesting to note the relationship between these three performance measures: when $\epsilon = 0$, $\min_p \Delta_p \times e(t) \le \mathbb{E}r(t) \le b \times e(t) \le b \times \ell(t)$, where the expectation in the regret is with respect to the random samples. As a result, any algorithm minimizing the worst case probability of error, $\ell(t)$, also controls the average probability of error, $e(t)$, and the simple regret $\mathbb{E}r(t)$. Note that the algorithms introduced in this chapter directly target the problem of minimizing $\ell(t)$.

3 Gap-based Exploration Algorithms

In this section, we describe two gap-based exploration algorithms (GapE and UGapEb) and show how they are implemented in the fixed-budget setting. While GapE, the first algorithm (with theoretical analysis) proposed for the multi-bandit case, aims at solving the multi-bandit (ϵ, m)-best arm identification problem for $\epsilon = 0$ and $m = 1$, UGapEb addresses the general case of ϵ and m.

• **The GapE algorithm:** Figure 5.3 contains the pseudo-code of the gap-based exploration (GapE) algorithm. At each time step t, the algorithm relies on the observations up to time $t-1$ to build an index $B_{pk}(t)$ for each bandit-arm pair, and then selects the pair $(q(t), I(t))$ with the highest index. The index B_{pk} consists of two terms. The first term is the negative of the estimated gap for arm k in bandit p. Similarly to other upper-confidence bound (UCB) methods (Auer et al., 2002), the second part, the confidence interval $\beta_k(t-1)$,[3] is an exploration term which forces the algorithm to pull arms that have been less explored. As a result, the algorithm tends to pull arms with small estimated gap and small number of pulls. The exploration parameter a tunes the level of exploration of the algorithm, and as it is shown by the theoretical analysis of Section 4, its optimal value depends on the *complexity* of the problem (see Section 4 for further discussion). At the end of round n, the forecaster returns for each bandit p

[3]Its shape will be discussed in more detail later in this section.

```
**UGapEb**$(\epsilon, m, n, a)$

  **Parameters:** accuracy $\epsilon$, number of arms $m$, budget $n$, exploration parameter $a$
  **Initialize:** Pull each bandit-arm pair $(p, k)$ once, update $\widehat{\mu}_{pk}(MK)$ and set $T_{pk}(MK) = 1$

  **for** $t = MK + 1, \dots, n$ **do**
    SELECT-ARM $(t)$
  **end for**

  Return $\Omega(n) = \arg\min_{J(t)} B_{J(t)}(t)$
```

```
SELECT-ARM$(t)$
  Compute $B_{pk}(t)$ for each bandit-arm pair $(p, k)$.
  Identify in each bandit $p$, the set of $m$ arms,
    $J_p(t) \in \arg\overset{1..m}{\min_{k \in A}} B_{pk}(t)$.
  Select the bandit $q(t) = \max_{p \in \{1,\dots,M\}} B_{J_p(t)}(t)$
  Pull the arm $I(t) = \mathrm{argmax}_{k \in \{l_t, u_t\}} \beta_k(t-1)$
  Observe $X_{q(t),I(t)}(T_{q(t),I(t)}(t-1)+1) \sim \nu_{q(t),I(t)}$
  Update $\widehat{\mu}_{q(t),I(t)}(t)$ and $T_{q(t),I(t)}(t)$
```

Figure 5.4: The pseudo-code for the UGapE algorithm in the fixed-budget setting (UGapEb) *(left)* and UGapE's arm-selection strategy *(right).*

the arm with the highest estimated mean, i.e., $J_p(n) = \mathrm{argmax}_k\, \widehat{\mu}_{pk}(n)$.

• **The UGapEb algorithm:** The unified gap-based exploration (UGapE) is a meta-algorithm designed to be implemented in both fixed-budget and fixed-confidence settings. In this section, we focus on the fixed budget setting, the UGapEb algorithm, whose pseudo-code is shown in Figure 5.4. At each time step t, UGapEb uses the arm-selection strategy, SELECT-ARM (see the left panel of Figure 5.4), an arm-selection strategy that also used by the fixed confidence variant of UGapE, the UGapEc algorithm.

At each time step t, UGapE first uses the observations up to time $t-1$ and computes an index for each bandit-arm pair (p, k),

$$B_{pk}(t) = \overset{m}{\max_{i \neq k}} U_{pi}(t) - L_{pk}(t) \tag{5.3}$$

where, $\forall t$, $\forall k \in A$, $\forall p \in \{1, \dots, M\}$,

$$U_{pk}(t) = \widehat{\mu}_{pk}(t-1) + \beta_{pk}(t-1) \quad , \quad L_{pk}(t) = \widehat{\mu}_{pk}(t-1) - \beta_{pk}(t-1). \tag{5.4}$$

In Equation 5.4, $\beta_{pk}(t-1)$ is a confidence interval,[4] and $U_{pk}(t)$ and $L_{pk}(t)$ are high probability upper and lower bounds on the mean of arm k of bandit p, μ_{pk}, after $t-1$ rounds. Note that the parameter a is used in the definition of the confidence interval β_{pk}, whose shape strictly depends on the concentration bound used by the algorithm. This will be discussed in more details latter in this section. From Equation 5.4, we may see that the index $B_{pk}(t)$ is an upper-bound on the simple regret r_{pk} of the kth

[4]To be more precise, $\beta_{pk}(t-1)$ is the width of a confidence interval or a confidence radius.

arm of bandit p (see Equation 5.1). We also define an index for a set S^p as $B_{S^p}(t) = \max_{i \in S^p} B_{pi}(t)$. Similarly to the arm index, B_S is also defined in order to upper-bound the simple regret r_{S^p} with high probability (see Lemma 5.1).

After computing the arm indices, UGapE finds a set of m arms $J_p(t)$ in each bandit p with minimum upper-bound on their simple regrets, i.e., $J_p(t) = \arg \overset{1..m}{\min_{k \in A}} B_{pk}(t)$. Therefore, we can define $B_{J_p(t)}(t) = \overset{m}{\min_{k \in A}} B_{pk}(t)$ as the upper bound on the simple regret of the set of arms $J_p(t)$ for each bandit p. UGapE then selects $q(t)$, the bandit with maximal upper bound on its simple regret, i.e., $q(t) = \max_{p \in \{1,\ldots,M\}} B_{J_p(t)}(t)$, to explore it. From $J_{q(t)}(t)$, it computes two arm indices $u_t = \text{argmax}_{j \notin J_{q(t)}(t)} U_j(t)$ and $l_t = \text{argmin}_{i \in J_{q(t)}(t)} L_i(t)$, where in both cases the tie is broken in favor of the arm with the largest uncertainty $\beta(t-1)$. Arms l_t and u_t are the worst possible arm among those in $J(t)$ and the best possible arm left outside $J_{q(t)}(t)$, respectively, and together they represent how bad the choice of $J_{q(t)}(t)$ could be. Finally, the algorithm selects and pulls the arm $I(t)$ as the arm with the larger $\beta(t-1)$ among u_t and l_t, observes a sample $X_{q(t),I(t)}\Big(T_{q(t),I(t)}(t-1)+1\Big)$ from the distribution $\nu_{q(t),I(t)}$, and updates the empirical mean $\widehat{\mu}_{q(t),I(t)}(t)$ and the number of pulls $T_{q(t),I(t)}(t)$ of the selected arm $I(t)$ in bandit $q(t)$. UGapEb returns, for each bandit p, the set of arms $J_p(t)$ with the smallest index, i.e., $\Omega_p(n) = \arg\min_{J_p(t)} B_{J_p(t)}(t),\ t \in \{1,\ldots,n\}$.

Note that our approach can be easily modified to tackle the problem of finding the set of (m_p, ϵ_p)-optimal arms in each bandit p, where the difference with respect to the previous version is that now ϵ and m depend on p. For this case, it is sufficient to select the bandit with maximal value of the quantity $B_{J_p(t)}(t) - \epsilon_p$ at each time t. This quantity is the difference between the upper-bound on the simple regret of bandit p and its maximum accepted simple regret.

• **Discussion on GapE and UGapEb:** In GapE and UGapEb, β_{pk} is the radius of the confidence interval. It can be derived for instance from the Chernoff-Hoeffding bound as

$$\beta_{pk}(t-1) = b\sqrt{\frac{a}{T_{pk}(t-1)}}. \tag{5.5}$$

In Section 4, we discuss how the parameter a can be tuned and we show that a should be tuned as a function of n and ϵ in UGapEb. Defining the confidence interval in a general form $\beta_{pk}(t-1)$ allows us to easily extend the algorithm by taking into account different (higher) moments of the arms (see Section 4.2.2 for the case of variance, where $\beta_{pk}(t-1)$ is obtained from the Bernstein inequality).

The indices used by GapE and UGapEb for an arm k in bandit p are both based on the estimation of the negative estimated gap $(-\widehat{\Delta}_{pk})$. While the index of GapE adds an exploratory term only related to arm k, UGapE also considers an exploratory term related to the other arms in bandit p that allows us to define $\widehat{\Delta}_{pk}$ (either u_t if $k \in J_p(t)$ or l_t if $k \notin J_p(t)$). GapE and UGapEb are conceptually similar algorithms. In fact, UGapE should be seen as an updated version of GapE, where the index can be understood as a high-probability upper bound on the simple regret incurred with respect to arm k (as proved in Lemma 5.1). Therefore, this greatly simplifies our analysis of UGapE with respect to the one of GapE. Moreover, UGapE is similar to the algorithm called LUCB (Kalyanakrishnan et al., 2012). They only differ in the definition of $J(t)$ and the fact that UGapE pulls the arm with the largest radius of the confidence interval among u_t and l_t while LUCB pulls both together. As shown in the Section 4.2.2, this latter difference permits UGapE-V (the version of GapE that takes into account the variance) to have better results in the case when the arms have different variances. LUCB was proposed to solve the best arm identification problem in single-bandit under the fixed confidence setting and with $m \geq 1$ and $\epsilon \geq 0$. In this chapter, we detail our analysis of UGapE in the multi-bandit case for both fixed budget and fixed confidence settings. The proof technique gives a simplified view on the GapE type of algorithms and is original as it is based on this new argument that the indices used by UGapE are upper bounds on the simple regret.

As Figures 5.3 and 5.4 indicate, both algorithms resembles the UCB-E algorithm (see Figure 2.5 in Audibert et al. 2010 for its pseudocode) designed to solve the pure exploration problem in the single-bandit setting with $\epsilon = 0$ and $m = 1$. Nonetheless, the use of the negative estimated gap, $-\widehat{\Delta}_{pk}$, instead of the estimated mean, $\widehat{\mu}_{pk}$, (used by UCB-E) is crucial in the multi-bandit setting. In the single-bandit problem with $m = 1$, since the best and second best arms have the same gap ($\Delta_{p(1)} = \min_{k \neq (1)_p} \Delta_{pk}$), GapE and UGapE consider them equivalent and tend to pull them the same number of times, while UCB-E tends to pull the best arm more often than the second best one. Despite this difference, the performance of both approaches in predicting the best arm after n pulls would be the same. This is due to the fact that the probability of error depends on the capability of the algorithm to distinguish between the optimal and sub-optimal arms, and this is not affected by a different allocation over the best and second best arms as long as the number of pulls allocated to that pair is large enough with respect to their gap. Despite this similarity, the two approaches become completely different in the multi-bandit case. In this case, if we run UCB-E on all the MK arms, it tends to pull more the arm with the highest mean over all the bandits, i.e., $k^* = \arg\max_{m,k} \mu_{mk}$. As a result, it would be accurate in predicting the best arm k^* over bandits, but may have an arbitrarily bad performance in predicting the best arm for each bandit, and thus, may incur a large error $\ell(n)$. On the other hand, GapE and UGapEb focus on the arms with the smallest gaps. This way, they assign more pulls to bandits, whose optimal arms are difficult to identify (i.e., bandits with arms with small gaps), and as will be shown in the next section, they achieve a high probability in identifying the best arm in each bandit.

4 Theoretical Analysis

In this section, we present high probability upper-bounds on the performance of GapE and UGapEb, introduced in Section 3. Our main focus is on the analysis of UGapEb. This analysis consists of two parts: the arm-selection strategy that will be discussed in Section 4.1.1 and the final performance bound for UGapEb that is the subject of

Section 4.1.2.

We define the complexity of the problem as

$$H_\epsilon = \sum_{p=1}^{M} \sum_{i=1}^{K} \frac{b^2}{\max(\frac{\Delta_{pi}+\epsilon}{2}, \epsilon)^2} \, . \tag{5.6}$$

Note that although the complexity has an explicit dependence on ϵ, it also depends on the number of arms m through the definition of the gaps Δ_{pi}, thus making it a complexity measure for the overall (ϵ, m) best arm identification problem.

We first report an upper-bound on the probability of error $\ell(n)$ for the GapE algorithm.

Theorem 5.1. *For $m = 1$ and $\epsilon = 0$, if we run GapE with parameter $0 < a \leq \frac{16}{9}\frac{n-MK}{H_0}$, then its probability of error satisfies*

$$\ell(n) \leq \mathbb{P}\Big(\exists p : J_p(n) \neq S^*_p\Big) \leq 2MKn \exp(-\frac{a}{64}),$$

in particular for $a = \frac{16}{9}\frac{n-MK}{H_0}$, we have $\ell(n) \leq 2MKn \exp(-\frac{1}{36}\frac{n-MK}{H_0})$.

We do not report the proof of this theorem in the thesis and refer the reader to Gabillon et al. (2011a). This is mainly because of the similarity between the proofs of GapE and UGapEb and the fact that GapE's proof is tedious compared to that of UGapEb. The main reason in turn for the latter is that the index of an arm in UGapEb is an upper bound on the regret associated with that arm, a property used extensively by the corresponding analysis, while the indices used in GapE does not possess the same property.

We now report a high probability bound on the regret of UGapEb:

Theorem 5.2. *If we run UGapEb with parameter $0 < a \leq \frac{n-K}{4H_\epsilon}$, its simple regret $r_{\Omega_p(n)}$ satisfies in each bandit p*

$$\widetilde{\delta} = \mathbb{P}\Big(r_{\Omega_p(n)} \geq \epsilon\Big) \leq 2MKn \exp(-2a),$$

in particular for $a = \frac{n-K}{4H_\epsilon}$, we have $\mathbb{P}\Big(r_{\Omega_p(n)} \geq \epsilon\Big) \leq 2MKn \exp(-\frac{1}{2}\frac{n-MK}{H_0})$.

Comparing the results of Theorems 5.1 and 5.2, it is easy to see that the bound on the performance of UGapEb generalizes the one for GapE to the (m, ϵ)-best arm identification, in addition to improving some numerical constants.

Remark 5.1 (The complexity terms). Theorems 5.1 and 5.2 indicate that the probability of success of GapE and UGapE are directly related to the complexity H_ϵ defined by Equation 5.6. H_ϵ captures the intrinsic difficulty of the (ϵ,m)-best arm(s) identification problem. Furthermore, note that this definition generalizes existing notions of complexity. For example, for $\epsilon = 0$ and $m = 1$, we recover the complexity used in the definition of UCB-E (Audibert et al., 2010) for the fixed budget setting. Let us analyze H_ϵ in the general case of $\epsilon > 0$. We define the complexity of a single arm $i \in A$ in bandit p, $H_{\epsilon,pi} = b^2/\max(\frac{\Delta_{pi}+\epsilon}{2}, \epsilon)^2$. When the gap Δ_{pi} is smaller than the desired accuracy ϵ, i.e., $\Delta_{pi} \leq \epsilon$, then the complexity reduces to $H_{\epsilon,pi} = 1/\epsilon^2$. In fact, the algorithm can stop as soon as the desired accuracy ϵ is achieved, which means that there is no need to exactly discriminate between arm i and the set of m best arms. On the other hand, when $\Delta_{pi} > \epsilon$, the complexity becomes $H_{\epsilon,pi} = 4b^2/(\Delta_{pi} + \epsilon)^2$. This shows that when the desired accuracy is smaller than the gap, the complexity of the problem is smaller than the case of $\epsilon = 0$, for which we have $H_{0,pi} = 4b^2/\Delta_{pi}^2$.

Remark 5.2 (Analysis of the bounds). If the time horizon (budget) n is known in advance, it would be possible to set the exploration parameter a as a linear function of n, and as a result, the probabilities of error of GapE and UGapEb decrease exponentially with the time horizon. The other interesting aspect of the bound is the complexity term H_ϵ appearing in the optimal value of the exploration parameter a (i.e., $a = \frac{4}{9}\frac{n-K}{H_0}$ for GapE and $a = \frac{n-K}{4H_\epsilon}$ for UGapEb). In all the following remarks, for simplicity, we consider $\epsilon = 0$ and drop the dependencies on ϵ for H. If we denote by $H_{pk} = b^2/\Delta_{pk}^2$, the complexity of arm k in bandit p, it is clear from the definition of H that each arm has an additive impact on the overall complexity of the multi-bandit problem. Moreover, if we define the complexity of each bandit p as $H_p = \sum_k b^2/\Delta_{pk}^2$ (similar to the definition of complexity for UCB-E in Audibert et al. 2010), the GapE and UGapEb complexities may be rewritten as $H = \sum_p H_p$. This means that the complexity of GapE and UGapEb is simply the sum of the complexities of all the bandits.

Remark 5.3 (Comparison with the static allocation strategy). The main objective of GapE and UGapEb is to trade-off between allocating pulls according to the gaps (more precisely, according to the complexities H_{pk}) and the exploration needed to improve the accuracy of their estimates. If the gaps were known in advance, a nearly-optimal static allocation strategy assigns to each bandit-arm pair a number of pulls proportional to its complexity. Let us consider a strategy that pulls each arm a fixed number of times over the horizon n. The probability of error for this strategy may be bounded as

$$\begin{aligned}\ell_{\text{Static}}(n) &\leq \mathbb{P}\Big(\exists p : J_p(n) \neq S_p^*\Big) \leq \sum_{p=1}^{M} \mathbb{P}\Big(J_p(n) \neq S_p^*\Big) \leq \sum_{p=1}^{M}\sum_{k\neq S_p^*} \mathbb{P}\Big(\widehat{\mu}_{p(m)}(n) \leq \widehat{\mu}_{pk}(n)\Big) \\ &\leq \sum_{p=1}^{M}\sum_{k\neq S_p^*} \exp\Big(-T_{pk}(n)\frac{\Delta_{pk}^2}{b^2}\Big) = \sum_{p=1}^{M}\sum_{k\neq S_p^*} \exp\Big(-T_{pk}(n)H_{pk}^{-1}\Big). \qquad (5.7)\end{aligned}$$

Given the constraint $\sum_{pk} T_{pk}(n) = n$, the allocation minimizing the last term in Equation 5.7 is $T_{pk}^*(n) = nH_{pk}/H$. We refer to this static allocation strategy as *StaticGap*. Although this is not necessarily the optimal static strategy ($T_{pk}^*(n)$ only minimizes an upper-bound), this allocation guarantees a probability of error smaller than $MK\exp(-n/H)$. Theorem 5.1 and Theorem 5.2 show that for n large enough, both GapE and UGapE achieve the same performance as the static allocation *StaticGap*.

Remark 5.4 (Comparison with SAR). Here we compare the bounds reported in Theorems 5.1 and 5.2 with the performance of the SAR algorithm (Wang et al., 2013).

SAR is an extension of the Successive Rejects algorithm (Audibert et al., 2010) to the multi-bandit setting. The budget n is divided into $MK-1$ phases. At the end of each phase, the arm with the biggest estimated gap is stopped from being pulled. During each phase all the remaining arms are pulled uniformly. The probability of error $\ell(n)$ of SAR may be bounded as

$$\ell_{\text{SAR}}(n) \leq \exp\Big(O\big(\frac{-n}{\overline{H} log(MK)}\big)\Big),$$

where $\overline{H} = \max_{i\in\{1,\dots,MK\}} \frac{i}{\Delta^2_{(i)}}$, with the gaps of all the arms in all the bandits ordered as $\Delta^2_{(1)} \leq \Delta^2_{(2)} \leq \dots \leq \Delta^2_{(MK)}$. As $\overline{H} \leq H \leq \overline{H}\log(2K)$, the theoretical results of SAR are slightly worse than those of (U)GapEb by a logarithmic factor in the exponent. On the other hand, the advantage of SAR is that it does not require the knowledge of the complexity, and thus, is parameter-free. In terms of practical comparison, note that experiments comparing GapE to SAR are reported in Wang et al. (2013), where they concluded that GapE outperforms SAR when the exploration parameter of GapE is well-tuned.

Remark 5.5 (Comparison with other allocation strategies). At the end of Section 3, we discussed the difference between GapE, UGapEb and UCB-E. Here we compare the bounds reported in Theorems 5.1 and 5.2 with the performance of the *Uniform* and the combined *Uniform+UCB-E* allocation strategies. In *Uniform*, the total budget n is uniformly split over all the bandits and arms. As a result, each bandit-arm pair is pulled $T_{pk}(n) = n/(MK)$ times. Using the same derivation as in Remark 5.3, the probability of error $\ell(n)$ for this strategy may be bounded as

$$\ell_{\text{Unif}}(n) \leq \sum_{p=1}^{M}\sum_{k\neq S^*_p} \exp\Big(-\frac{n}{MK}\frac{\Delta^2_{pk}}{b^2}\Big) \leq MK\exp\Big(-\frac{n}{MK\max_{p,k} H_{pk}}\Big).$$

In the *Uniform+UCB-E* allocation strategy, i.e., a two-level algorithm that first selects a bandit uniformly and then pulls arms within each bandit using UCB-E, the total number of pulls for each bandit p is $\sum_k T_{pk}(n) = n/M$, while the number of pulls $T_{pk}(n)$ over the arms in bandit p is determined by UCB-E. Thus, the probability of error of this strategy may be bounded as

$$\ell_{\text{Unif+UCB-E}}(n) \leq \sum_{p=1}^{M} 2nK\exp\Big(-\frac{n/M-K}{18H_p}\Big) \leq 2nMK\exp\Big(-\frac{n/M-K}{18\max_p H_p}\Big),$$

where the first inequality follows from Theorem 1 in Audibert et al. (2010) (recall that $H_p = \sum_k b^2/\Delta^2_{pk}$). Let $b=1$ (i.e., all the arms have distributions bounded in $[0,1]$), up to constants and multiplicative factors in front of the exponentials, and if n is large enough compared to M and K (so as to approximate $n/M-K$ and $n-K$ by n), the probability of error for the four algorithms (the one of UGapEb is similar to the one of GapE) may be bounded as

$$\ell_{\text{Unif}}(n) \leq \exp\left(O\Big(\frac{-n/MK}{\max\limits_{p,k} H_{pk}}\Big)\right), \qquad \ell_{\text{U+UCBE}}(n) \leq \exp\left(O\Big(\frac{-n/M}{\max\limits_{p} H_{p}}\Big)\right),$$

$$\ell_{\text{UGapEb}}(n) \simeq \ell_{\text{GapE}}(n) \leq \exp\left(O\Big(\frac{-n}{\sum\limits_{p,k} H_{pk}}\Big)\right).$$

By comparing the arguments of the exponential terms, we have the trivial sequence of inequalities $MK \max_{p,k} H_{pk} \geq M \max_p \sum_k H_{pk} \geq \sum_{p,k} H_{pk}$, which implies that the upper bound on the probability of error of GapE and UGapEb are usually significantly smaller. This relationship, which is confirmed by the experiments reported in Section 5, shows that GapE and UGapEb are able to adapt to the complexity H of the overall multi-bandit problem better than the other two allocation strategies. In fact, while the performance of the *Uniform* strategy depends on the most *complex* arm over the bandits and the strategy *Unif+UCB-E* is affected by the most complex bandit, the performance of GapE and UGapEb depends on the sum of the complexities of all the arms involved in the problem.

4.1 Proof of Theorem 5.2

Before moving to the detailed analysis, we define additional notation used in the analysis. We first define event $\mathcal{E}$ as

$$\mathcal{E} = \Big\{\forall k \in A,\ \forall p \in \{1,\dots,M\},\ \forall t \in \{1,\dots,n\},\ \Big|\widehat{\mu}_{pk}(t) - \mu_{pk}\Big| < \beta_{pk}(t)\Big\}. \tag{5.8}$$

Note that event $\mathcal{E}$ plays an important role in the sequel, i.e., when $\mathcal{E}$ holds, we have that for any arm $k \in A$ in any bandit p and at any time t, $L_{pk}(t) \leq \mu_{pk} \leq U_{pk}(t)$.

4.1.1 Analysis of the Arm-Selection Strategy

Here we report lower (Lemma 5.1) and upper (Lemma 5.4) bounds for indices B_S on event $\mathcal{E}$, which show their connection with the regret and gaps. We also report two technical lemmas used in the proofs (Lemmas 5.2 and 5.3 and Corollary 5.1). We first prove that for any bandit p, any set $S \neq S^*_p$, and any time $t \in \{1,\dots,n\}$, the index $B_S(t)$ is an upper-bound on the simple regret of this set r_S.

Lemma 5.1. *On event $\mathcal{E}$, in any bandit p, for any set $S \neq S^*_p$, and any time $t \in \{1,\dots,n\}$, we have $B_S(t) \geq r_S$.*

Proof. On event $\mathcal{E}$, in any bandit p, for any arm $i \notin S^*_p$, and at each time $t \in \{1,\dots,n\}$, we may write

$$\begin{aligned} B_{pi}(t) &= \overset{m}{\max_{j\neq i}} U_{pj}(t) - L_{pi}(t) = \overset{m}{\max_{j\neq i}} \Big(\widehat{\mu}_{pj}(t-1) + \beta_{pj}(t-1)\Big) - \Big(\widehat{\mu}_{pi}(t-1) - \beta_{pi}(t-1)\Big) \\ &\geq \overset{m}{\max_{j\neq i}} \mu_{pj} - \mu_{pi} = \mu_{(m)} - \mu_{pi} = r_{pi}\ . \end{aligned} \tag{5.9}$$

Using Equation 5.9, we have

$$B_S(t) = \max_{i\in S} B_{pi}(t) \geq \max_{i\in(S-S_p^*)} B_{pi}(t) \geq \max_{i\in(S-S_p^*)} r_{pi} = r_S,$$

where the last passage follows from the fact that $r_{pi} \leq 0$ for any $i \in S_p^*$. $\square$

We define $B(t)$ for each time $t \in \{1,\dots,n\}$ as

$$B(t) = U_{u_t}(t) - L_{l_t}(t) = \max_{j\notin J(t)} U_j(t) - \min_{i\in J(t)} L_i(t), \tag{5.10}$$

and prove the following lemma for this quantity.

Lemma 5.2. *For each $t \in \{1,\dots,n\}$ and for all bandit p, we have $B_{J_p(t)}(t) \leq B(t)$.*

Proof. We first focus on the bandit in which an arm is pulled at time t, $q(t)$. For $t \in \{1,\dots,n\}$, we may write

$$\begin{aligned} B_{J_{q(t)}(t)}(t) &= \max_{i\in J_{q(t)}(t)} \left(\overset{m}{\max_{j\neq i}} U_{q(t)j}(t) - L_{q(t)i}(t) \right) \leq \max_{i\in J_{q(t)}(t)} \overset{m}{\max_{j\neq i}} U_{q(t)j}(t) - \min_{i\in J_{q(t)}(t)} L_{q(t)i}(t) \\ &\overset{(a)}{\leq} \max_{j\notin J_{q(t)}(t)} U_{q(t)j}(t) - \min_{i\in J(t)} L_{q(t)i}(t) = U_{u_t}(t) - L_{l_t}(t) = B(t). \end{aligned}$$

(a) Let set $Z = \max_{i\in J_{q(t)}(t)} \overset{m}{\max_{j\neq i}} U_{q(t)j}(t)$ and define $\widehat{S}(t) = \arg \overset{1..m}{\max_{i\in A}} U_{q(t)i}(t)$ to be the set of m arms with the largest $U(t)$. If $J_{q(t)}(t) = \widehat{S}(t)$ then $Z = U_{q(t)(m+1)}(t) = \max_{j\notin J_{q(t)}(t)} U_{q(t)j}(t)$, and if $J - q(t)(t) \neq \widehat{S}(t)$ then $Z = U_{q(t)(m)}(t) \leq \max_{j\notin J_{q(t)}(t)} U_{q(t)j}(t)$.

Finally, we have by definition that for all bandit p, $B_{J_{q(t)}(t)}(t) \geq B_{J_p(t)}(t)$. $\square$

Lemma 5.3. *At each time $t \in \{1,\dots,n\}$, we have that*[5]

$$\begin{aligned} &\textit{if } u_t \textit{ is pulled,} \qquad L_{u_t}(t) \leq L_{l_t}(t) && (5.11)\\ &\textit{if } l_t \textit{ is pulled,} \qquad U_{u_t}(t) \leq U_{l_t}(t). && (5.12) \end{aligned}$$

Proof. We consider the following two cases:

Case 1. $\widehat{\mu}_{u_t}(t-1) \leq \widehat{\mu}_{l_t}(t-1)$**:** If we pull u_t, by definition we have $\beta_{u_t}(t-1) \geq \beta_{l_t}(t-1)$, and thus, Equation 5.11 holds. A similar reasoning gives Equation 5.12 when l_t is pulled.

Case 2. $\widehat{\mu}_{u_t}(t-1) > \widehat{\mu}_{l_t}(t-1)$**:** We consider the following two sub-cases:

Case 2.1. u_t is pulled**:** We prove this case by contradiction. Let us assume that

[5]We recently noticed that our analysis could be also applied to the LUCB algorithm (Kalyanakrishnan et al., 2012), an algorithm with similar structure to UGapE. In fact, this lemma becomes even simpler to prove for the LUCB algorithm.

$L_{u_t}(t) > L_{l_t}(t)$. Since arm u_t is pulled, by definition we have $\beta_{u_t}(t-1) \geq \beta_{l_t}(t-1)$, and thus, $U_{u_t}(t) > U_{l_t}(t)$. From this, it is easy to see that $\max^m_{j \neq l_t} U_{q(t)j}(t) \geq \max^m_{j \neq u_t} U_{q(t)j}(t)$. Using these results, we may write

$$B_{u_t}(t) = \max^m_{j \neq u_t} U_{q(t)j}(t) - L_{u_t}(t) < \max^m_{j \neq l_t} U_{q(t)j}(t) - L_{l_t}(t) = B_{l_t}(t).$$

Since $u_t \notin J(t)$ and $l_t \in J_{q(t)}(t)$ and $J_{q(t)}(t)$ collects all the arms with the smallest $B_{q(t)i}$ values in bandit $q(t)$, we have $B_{u_t}(t) \geq B_{l_t}(t)$ by definition. This contradicts the previous statement, and thus, Equation 5.11 holds.

Case 2.2. l_t is pulled**:** With a similar reasoning to Case 2.1. and by contradiction, we can show that Equation 5.12 holds in this case. □

Corollary 5.1. *If arm k is pulled at time $t \in \{1, \ldots, n\}$, then we have $B(t) \leq 2\beta_{pk}(t-1)$.*

Proof. The proof is a direct application of Lemma 5.3. We know that the pulled arm k in bandit p is either u_t or l_t. If $k = u_t$, we have

$$B(t) = U_{u_t}(t) - L_{l_t}(t) \overset{\text{(a)}}{\leq} U_{u_t}(t) - L_{u_t}(t) = 2\beta_{u_t}(t-1) = 2\beta_{q(t)k}(t-1),$$

where **(a)** is from Equation 5.11 in Lemma 5.3. Similarly if $k = l_t$, we have

$$B(t) = U_{u_t}(t) - L_{l_t}(t) \overset{\text{(b)}}{\leq} U_{l_t}(t) - L_{l_t}(t) = 2\beta_{l_t}(t-1) = 2\beta_{q(t)k}(t-1),$$

where **(b)** is from Equation 5.12 in Lemma 5.3. □

Lemma 5.4. *On event $\mathcal{E}$, if arm $k \in \{l_t, u_t\}$ is pulled at time $t \in \{1, \ldots, n\}$, we have*

$$B_{J_{q(t)}(t)}(t) \leq \min\Big(0, -\Delta_{q(t)k} + 2\beta_{q(t)k}(t-1)\Big) + 2\beta_{q(t)k}(t-1). \tag{5.13}$$

Proof. We first prove the statement for $B(t) = U_{u_t}(t) - L_{l_t}(t)$, i.e.,

$$B(t) \leq \min\Big(0, -\Delta_{q(t)k} + 2\beta_{q(t)k}(t-1)\Big) + 2\beta_{q(t)k}(t-1). \tag{5.14}$$

We consider the following cases:
Case 1. $k = u_t$**:**
Case 1.1. $u_t \in S^*_{q(t)}$**:** Since by definition $u_t \notin J_{q(t)}(t)$, there exists an arm $j \notin S^*_{q(t)}$ such that $j \in J_{q(t)}(t)$. Now we may write

$$\begin{aligned}\mu_{q(t)(m+1)} \geq \mu_{q(t)j} \overset{\text{(a)}}{\geq} L_{q(t)j}(t) \overset{\text{(b)}}{\geq} L_{l_t}(t) \overset{\text{(c)}}{\geq} L_{u_t}(t) &= \widehat{\mu}_{q(t)k}(t-1) - \beta_{q(t)k}(t-1) \\ &\overset{\text{(d)}}{\geq} \mu_{q(t)k} - 2\beta_{q(t)k}(t-1)\end{aligned} \tag{5.15}$$

(a) and **(d)** hold because of event $\mathcal{E}$, **(b)** follows from the fact that $j \in J_{q(t)}(t)$ and from the definition of l_t, and **(c)** is the result of Lemma 5.3. From Equation 5.15, we may deduce that $-\Delta_{q(t)k} + 2\beta_{q(t)k}(t-1) \geq 0$, which together with Corollary 5.1 give us the desired result (Equation 5.14).

Case 1.2. $u_t \notin S^*_{q(t)}$**:**
Case 1.2.1. $l_t \in S^*_{q(t)}$**:** In this case, we may write

$$\begin{aligned} B(t) = U_{u_t}(t) - L_{l_t}(t) &\overset{\text{(a)}}{\leq} \mu_{u_t} + 2\beta_{u_t}(t-1) - \mu_{l_t} + 2\beta_{l_t}(t-1) \\ &\overset{\text{(b)}}{\leq} \mu_{u_t} + 2\beta_{u_t}(t-1) - \mu_{q(t)(m)} + 2\beta_{l_t}(t-1) \overset{\text{(c)}}{\leq} -\Delta_{u_t} + 4\beta_{u_t}(t-1) \end{aligned} \tag{5.16}$$

(a) holds because of event $\mathcal{E}$, **(b)** is from the fact that $l_t \in S^*_{q(t)}$, and **(c)** is because u_t is pulled, and thus, $\beta_{u_t}(t-1) \geq \beta_{l_t}(t-1)$. The final result follows from Equation 5.16 and Corollary 5.1.

Case 1.2.2. $l_t \notin S^*_{q(t)}$**:** Since $l_t \notin S^*_{q(t)}$ and the fact that by definition $l_t \in J_{q(t)}(t)$, there exists an arm $j \in S^*_{q(t)}$ such that $j \notin J_{q(t)}(t)$. Now we may write

$$\mu_{u_t} + 2\beta_{u_t}(t-1) \overset{\text{(a)}}{\geq} U_{u_t}(t) \overset{\text{(b)}}{\geq} U_{q(t)j}(t) \overset{\text{(c)}}{\geq} \mu_j \overset{\text{(d)}}{\geq} \mu_{q(t)(m)} \tag{5.17}$$

(a) and **(c)** hold because of event $\mathcal{E}$, **(b)** is from the definition of u_t and the fact that $j \notin J_{q(t)}(t)$, and **(d)** holds because $j \in S^*_{q(t)}$. From Equation 5.17, we may deduce that $-\Delta_{u_t} + 2\beta_{u_t}(t-1) \geq 0$, which together with Corollary 5.1 give us the final result (Equation 5.14).

With similar arguments and cases, we prove the result of Equation 5.14 for $k = l_t$. The final statement of the lemma (Equation 5.13) follows directly from $B_{J_{q(t)}(t)}(t) \geq B(t)$ as shown in Lemma 5.2. □

Using Lemmas 5.1 and 5.4, we define an upper and a lower bounds on $B_{J_{q(t)}(t)}$ in terms of quantities related to the regret of $J_{q(t)}(t)$. Lemma 5.1 confirms the intuition that the B-values upper-bound the regret of the corresponding set of arms (with high probability). Unfortunately, this is not enough to claim that selecting $J_p(t)$ in bandit p as the set of arms with smallest B-values actually correspond to arms with small regret, since $B_{J_p(t)}$ could be an arbitrary loose bound on the regret. Lemma 5.4 provides this complementary guarantee specifically for the set $J_{q(t)}(t)$, in the form of an upper-bound on $B_{J_{q(t)}(t)}$ with respect to the gap of $k \in \{u_t, l_t\}$. This implies that as the algorithm runs, the choice of $J_{q(t)}(t)$ becomes more and more accurate since $B_{J_{q(t)}(t)}$ is constrained between $r_{J_{q(t)}(t)}$ and a quantity (Equation 5.13) that gets smaller and smaller, thus implying that selecting the arms with the smaller B-value, i.e., the set $J_{q(t)}(t)$, corresponds to those which actually have the smallest regret, i.e., the arms in $S^*_{q(t)}$. Moreover, this observation will be true among all the bandits as the UGapEb algorithm always select the bandit with the highest B index. These arguments will be implicitly at the basis of the proof of the following theorem.

4.1.2 Proof of the Regret Bound

Here we prove an upper-bound on the simple-regret of UGapEb. From the definition of the confidence interval $\beta_{pi}(t)$ in Equation 5.5 and a union bound, we have that $\mathbb{P}(\mathcal{E}) \geq 1 - 2Kn\exp(-2a)$.[6] We now have all the tools needed to prove the performance of UGapEb for the multi-bandit m (ϵ,m)-best arm identification problem.

Proof. The proof is by contradiction. We assume that their exist a bandit p for which $r_{\Omega_p(n)} > \epsilon$ on event $\mathcal{E}$ and consider the following two steps:

Step 1: Here we show that on event $\mathcal{E}$, we have the following upper-bound on the number of pulls of any arm $i \in A$:

$$T_{pi}(n) < \frac{4ab^2}{\max\left(\frac{\Delta_{pi}+\epsilon}{2}, \epsilon\right)^2} + 1. \tag{5.18}$$

Let t_{pi} be the last time that arm i in bandit p is pulled. If arm i has been pulled only during the initialization phase, $T_{pi}(n) = 1$ and Equation 5.18 trivially holds. If i has been selected by SELECT-ARM, then we have

$$\min\Big(-\Delta_{pi} + 2\beta_{pi}(t_{pi}-1), 0\Big) + 2\beta_{pi}(t_{pi}-1) \overset{\text{(a)}}{\geq} B(t_{pi}) \overset{\text{(b)}}{\geq} B_{J(t_{pi})}(t_{pi}) \overset{\text{(c)}}{\geq} B_{\Omega(n)}(t_{\ell_p}) \overset{\text{(d)}}{>} \epsilon, \tag{5.19}$$

where $t_{\ell_p} \in \{1, \ldots, n\}$ is the time such that $\Omega_p(n) = J(t_{\ell_p})$. **(a)** and **(b)** are the results of Lemmas 5.4 and 5.2, **(c)** is by the definition of $\Omega_p(n)$, and **(d)** holds because using Lemma 5.1, we know that if the algorithm suffers a simple regret $r_{\Omega_p(n)} > \epsilon$ (as assumed at the beginning of the proof), then $\forall t = 1, \ldots, n+1$, $B_{\Omega_p(n)}(t) > \epsilon$. By the definition of t_i, we know $T_{pi}(n) = T_{pi}(t_i - 1) + 1$. Using this fact, the definition of $\beta_{pi}(t_i - 1)$, and Equation 5.19, it is straightforward to show that Equation 5.18 holds.

Step 2: We know that $\sum_{p=1}^{M}\sum_{i=1}^{K} T_{pi}(n) = n$. Using Equation 5.18, we have $\sum_{p=1}^{M}\sum_{i=1}^{K} \frac{4ab^2}{\max\left(\frac{\Delta_{pi}+\epsilon}{2},\epsilon\right)^2} + K > n$ on event $\mathcal{E}$. It is easy to see that by selecting $a \leq \frac{n-K}{4H_\epsilon}$, the left-hand-side of this inequality will be smaller than or equal to n, which is a contradiction. Thus, we conclude that $r_{\Omega_p(n)} \leq \epsilon$ on event $\mathcal{E}$. The final result follows from the probability of event $\mathcal{E}$ defined at the beginning of this section. □

4.2 Extensions

In this section, we propose variants of the UGapE and GapE algorithms with the objective of extending their applicability and improving their performance.

[6]The extension to a confidence interval that takes into account the variance of the arms is discussed in Section 4.2.2

4.2.1 Extension to the Fixed-Confidence Setting

Here we give a variant of the UGapE algorithm, called UGapEc, for the fixed confidence setting. The pseudo-code of UGapEc is displayed in Figure 5.5. This algorithm is mainly based on the same sampling routine as UGapEb defined in the left panel of Figure 5.4. Therefore, its analysis will also mainly reuse the results from the analysis of the sampling routine reported in Section 4.1.1.

There are two more points that need to be discussed on UGapEc in comparison with UGapEb. **1)** While UGapEc defines the set of returned arms for bandit p as $\Omega_p(t) = J_p(t)$, UGapEb returns the set of arms $J_p(t)$ with the smallest index, i.e., $\Omega_p(n) = \arg\min_{J(t)} B_{J_p(t)}(t),\ t \in \{1, \ldots, n\}$. **2)** UGapEc stops (we refer to the number of rounds before stopping as $\widetilde{n}$) when $B_{J_p(\widetilde{n}+1)}(\widetilde{n}+1)$ is less than the given accuracy ϵ for all bandits p, i.e., when even the mth worst upper-bound on the arm simple regret among all the arms in the selected set $J_p(\widetilde{n}+1)$ is smaller than ϵ in all the bandits. This guarantees that the simple regret (see Equation 5.2) of the set returned by the algorithm, $\Omega_p(\widetilde{n}) = J(\widetilde{n}+1)$, to be smaller than ϵ with probability larger than $1-\delta$.

```
UGapEc ($\epsilon, m, \delta, c$)
  Parameters: accuracy $\epsilon$, number of arms $m$, confidence level $\delta$, exploration parameter $c$
  Initialize: Pull each arm $k$ once in each bandit $p$, update $\widehat{\mu}_{pk}(MK)$, set $T_{pk}(MK) = 1$ and $t \leftarrow MK + 1$

  while $\exists p: B_{J_p(t)}(t) \geq \epsilon$ do
    SELECT-ARM ($t$)
    $t \leftarrow t + 1$
  end while

  For all bandit $p$, return $\Omega_p(t) = J_p(t)$
```

Figure 5.5: The pseudo-code of UGapEc, the UGapE algorithm in the fixed-confidence setting.

Since the setting considered by the algorithm is fixed-confidence, we do not consider a fixed stopping time n. Therefore, similar to event $\mathcal{E}$, we define event $\mathcal{E}_c$ as

$$\mathcal{E}_c = \Big\{\forall k \in A,\ \forall p \in \{1, \ldots, M\},\ \forall t \in \{1, \ldots, +\infty\},\ \Big|\widehat{\mu}_{pk}(t) - \mu_{pk}\Big| < \beta_{pk}(t)\Big\}, \quad (5.20)$$

where $\beta_{pi}(t)$ takes the following form,

$$\beta_{pi}(t-1) = b\sqrt{\frac{c \log \frac{4MK(t-1)^3}{\delta}}{T_{pi}(t-1)}}\ . \quad (5.21)$$

From the definition of the confidence interval $\beta_{pi}(t)$ in Equation 5.5 and a union bound on $T_{pk}(t) \in \{0, \ldots, t\},\ t = 1, \ldots, \infty$, we have that $\mathbb{P}(\mathcal{E}_c) \geq 1 - \delta$. In the next theorem, we prove an upper-bound on the simple-regret of UGapEc.

Theorem 5.3. *The UGapEc algorithm stops after $\tilde{n}$ rounds and returns a set of m arms in each bandit p, $\Omega_p(\tilde{n})$, that for each bandit p satisfies*

$$\mathbb{P}\Big(r_{\Omega_p(\tilde{n}+1)} \leq \epsilon \wedge \tilde{n} \leq N\Big) \geq 1 - \delta,$$

where $N = K + O(H_\epsilon \log \frac{H_\epsilon MK}{\delta})$ and c has been set to its optimal value $1/2$.

Proof. We first prove the bound on the simple regret of UGapEc. Using Lemma 5.1, we have that on event $\mathcal{E}_c$, for each bandit p, the simple regret of UGapEc upon stopping satisfies $B_{J_p(\tilde{n}+1)}(\tilde{n}+1) = B_{\Omega_p(\tilde{n}+1)}(\tilde{n}+1) \geq r_{\Omega_p(\tilde{n}+1)}$. As a result, on event $\mathcal{E}_c$, the regret of UGapEc cannot be bigger than ϵ, because then it contradicts the stopping condition of the algorithm, i.e., $B_{J_p(\tilde{n}+1)}(\tilde{n}+1) < \epsilon$ for each bandit p. Therefore, we have $\mathbb{P}\Big(r_{\Omega_p(\tilde{n}+1)} \leq \epsilon\Big) \geq 1 - \delta$ for all bandits. Now we prove the bound for the sample complexity of UGapEc. Similar to the proof of Theorem 5.2, we consider the following two steps:

Step 1: Here we show that on event $\mathcal{E}_c$, we have the following upper-bound on the number of pulls of any arm $i \in \mathcal{A}$ in bandit p:

$$T_{pi}(\tilde{n}) \leq \frac{2b^2 \log(4MK(\tilde{n}-1)^3/\delta)}{\max\left(\frac{\Delta_{pi+\epsilon}}{2}, \epsilon\right)^2} + 1. \tag{5.22}$$

Let t_{pi} be the last time that arm i is pulled in bandit p. If arm i in bandit p has been pulled only during the initialization phase, $T_{pi}(\tilde{n}) = 1$ and Equation 5.22 trivially holds. If i in p has been selected by SELECT-ARM, then we have $B_{J_p(t_{pi})}(t_{pi}) \geq \epsilon$. Now using Lemma 5.4, we may write

$$B_{J_p(t_{pi})}(t_{pi}) \leq \min\Big(0, -\Delta_{pi} + 2\beta_{pi}(t_{pi}-1)\Big) + 2\beta_{pi}(t_{pi}-1). \tag{5.23}$$

We can prove Equation 5.22 by plugging in the value of $\beta_{pi}(t_{pi}-1)$ from Equation 5.5 and solving Equation 5.23 for $T_{pi}(t_{pi})$ taking into account that $T_{pi}(t_{pi}-1)+1 = T_{pi}(t_{pi})$.

Step 2: We know that $\sum_{p=1}^{M}\sum_{i=1}^{K} T_{pi}(\tilde{n}) = \tilde{n}$. Using Equation 5.22, on event $\mathcal{E}_c$, we have $2H_\epsilon \log\Big(MK(\tilde{n}-1)^3/\delta\Big) + MK \geq \tilde{n}$. Using Lemma 8 in Antos et al. (2010) solves this inequality and gives us $\tilde{n} \leq N$. □

Theorem 5.3 indicates that similar to UGapEb, both the probability of success and sample complexity of UGapEc are directly related to the complexity H_ϵ defined by Equation 5.6. Furthermore, note that this definition generalizes existing notions of complexity. For example, for $\epsilon = 0$ and $m = 1$, we recover the complexity defined by Even-Dar et al. (2006) for the fixed accuracy setting. Moreover, the analysis reported here suggests that the performance of an upper confidence bound based algorithm such as UGapE is characterized by the same notion of complexity in both fixed budget and fixed confidence settings. Thus, whenever the complexity is known, it is possible to exploit the theoretical analysis (bounds on the performance) to easily switch from one setting to another. For example, as also suggested in Section 5.4 of Kalyanakrishnan (2011), if the complexity H is known, an algorithm like UGapEc can be adapted to run in the fixed budget setting by inverting the bound on its sample complexity. This would lead to an algorithm similar to UGapEb, with similar performance. A more intuitive way to see the link between the fixed confidence setting and the fixed budget setting with known complexity is to look at the "good" static allocation associated with these two settings. As derived in Equation 5.7, the number of pulls of the static allocation in the fixed budget setting is $T^*_{pk}(n) \approx nH_{pk}/H$ for arm k in bandit p. Similarly one can derive the static allocation in the fixed confidence setting for the same arm as $T^*_{pk}(n) \approx H_{pk}\log(\frac{1}{\delta})$. Hence, when the complexity H of the problem is known, one can turn an algorithm designed to mimic the "good" static allocation in the fixed confidence setting to an algorithm designed to mimic the "good" static allocation in the fixed budget setting by selecting the targeted accuracy so that $\log(\frac{1}{\delta}) \approx n/H$. On the other hand, it is an open question whether it is possible to find an "equivalence" between the two settings when the complexity is not known.

Note that contrary to this unifying approach, Karnin et al. (2013) have recently designed new algorithms for each setting separately (tackling the fixed budget setting without knowledge of the complexity) with improved, and almost optimal, new upper bounds on the probability of error of the proposed algorithms.

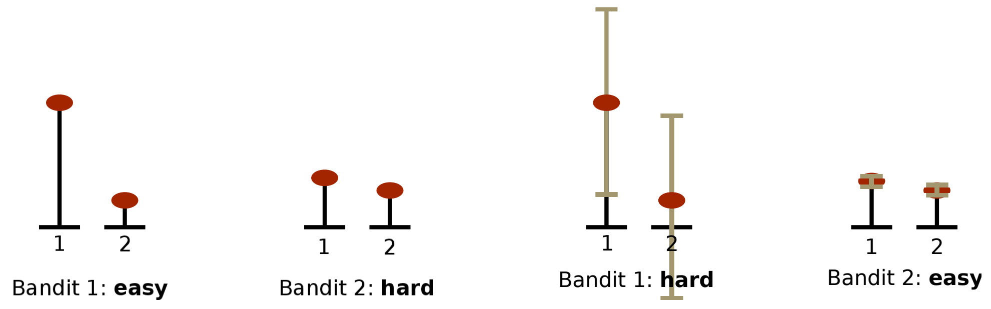
4.2.2 GapE-Variance and UGapE-Variance

In this section, we extend UGapE algorithms so that they take into account the variance of the arms and show how the resulting algorithms improve upon some previous approaches that were also taking the variance into account. As discussed in Section 3, UGapE pulls arms according to their B index, which is a high probability upper-bound on their simple regret. This gives us the flexibility of using any high probability bound in the definition of index in UGapE and to extend the algorithm. As discussed earlier, the algorithm and analysis (for both settings) are also modular enough to allow such extension. One natural extension, similar to the extension of UCB proposed by Audibert et al. (2007) for the cumulative regret setting, is to replace the Chernoff-Hoeffding bounds of Equation 5.5 with the following Bernstein bounds in order to take into account the variances of the arms:

$$\text{UGapEb-V:} \qquad \beta_{pk}(t) = \sqrt{\frac{2a\,\widehat{\sigma}^2_{pk}(t)}{T_{pk}(t)}} + \frac{(7/3)ab}{T_{pk}(t)-1},$$

$$\text{UGapEc-V:} \qquad \beta_{pk}(t) = \sqrt{\frac{2c\log\frac{Kt^3}{\delta}\,\widehat{\sigma}^2_{pk}(t)}{T_{pk}(t)}} + \frac{(7/3)bc\log\frac{Kt^3}{\delta}}{T_{pk}(t)-1},$$

where $\widehat{\sigma}^2_{pk}(t) = \frac{1}{T_{pk}(t)-1}\sum_{s=1}^{T_{pk}(t)}\Big(X_{pk}(s) - \widehat{\mu}_{pk}(t)\Big)^2$ is the estimated variance of arm k of bandit p at the end of round t. We call the resulting algorithm UGapE-variance (UGapE-V). Using Theorem 11 in Maurer and Pontil (2009), it is easy to show that Theorems 5.2 and 5.3, bounding the simple regret of the fixed budget and fixed confidence settings, still hold (without major change in their proofs) with a new definition



(a) Not taking into account the variance

(b) Taking into account the variance

Figure 5.6: On the importance of taking the variance into account. We consider a problem with two bandits both composed of two arms ($M = 2, K = 2$). In Bandit 1, the gap between the two arms is larger than in Bandit 2. On the left panel (a), if we do not take into account the variances of the arms, we will conclude that Bandit 1 is simpler to solve than Bandit 2. If it happens, as illustrated on the right panel (b), that the variance of the arms of Bandit 1 are way larger than the variance of the arms of Bandit 2, we would conclude, on the contrary, that Bandit 1 is more complex to solve that Bandit 2. Therefore in this example, an algorithm taking into account the gaps but not the variances would allocate most of its pulls to the simpler bandit and would subsequently perform poorly.

of complexity, i.e.,

$$H_\epsilon^\sigma = \sum_{p=1}^{M} \sum_{i=1}^{K} \frac{\left(\sigma_i + \sqrt{(13/3)b\Delta_{pi}}\right)^2}{\max\left(\frac{\Delta_{pi}+\epsilon}{2}, \epsilon\right)^2}. \tag{5.24}$$

This variance-complexity H_ϵ^σ is expected to better capture the complexity of the arms and to be smaller than H_ϵ defined by Equation 5.6, whenever the variances of the arms are small compared to the range b of the distribution. Figure 5.6 gives an example where taking the variance into account is crucial.

Mnih et al. (2008) proposed the Bernstein Races, a family of best arm identification algorithms based on the Bernstein inequality for the fixed confidence setting. The term bounding the number of pulls of a sub-optimal arm i in bandit p in their analysis is of the form $(\sigma_{p(1)}^2 + \sigma_{pi}^2)/\Delta_{pi}^2$, where $\sigma_{p(1)}^2$ is the variance of the best arm. This causes Bernstein Races to allocate the pulls equally over the arms, when the task is to discriminate between two arms ($K = 2$), while intuitively the arms should be pulled proportionally to their variances. On the other hand, UGapEc-V is able to handle this case properly (i.e., to pull the arms proportionally to their variances), because its bound on the number of pulls of a sub-optimal arm i in bandit p is of the form $\sigma_{pi}^2/\Delta_{pi}^2$ (see the definition of H_ϵ^σ in Equation 5.24). In Section 5.2.1, we report numerical results showing that UGapEc-V has better performance than Bernstein Race when variance of the optimal arm is larger than those of the sub-optimal arms.

A very similar variance extension can be proposed for GapE resulting in the GapE-

Variance (GapE-V) algorithm with the price of a separate and tedious proof (see Gabillon et al. 2011a). Naturally, the variance complexity of GapE is the same as the one of UGapEb for the case $\epsilon = 0$, but with worse numerical constants.

4.2.3 Higher-Order Moments

After considering the extension using variance estimates, a natural extension is to build algorithms taking into account higher-order moments. In fact, when considering all the moments of a distribution, the natural quantity for discriminating between the arms is related to the Kullback-Leibler (KL) divergence. The KL divergence between two continuous distributions ν_1 and ν_2, $D_{\mathrm{KL}}(\nu_1\|\nu_2)$, is defined as follows:

$$D_{\mathrm{KL}}(\nu_1\|\nu_2) = \int_{-\infty}^{\infty} \ln\left(\frac{\nu_1(x)}{\nu_2(x)}\right)\nu_1(x)\,\mathrm{d}x. \tag{5.25}$$

Intuitively, $D_{\mathrm{KL}}(\nu_1\|\nu_2)$ is a measure of the information loss when ν_2 is used to approximate ν_1 and is related to the number of pulls needed to discriminate between ν_1 and ν_2. Note that the KL divergence is not symmetric. The KL divergence has been used in the analysis and design of various algorithms for the classical (cumulative regret) bandit problem (Honda and Takemura, 2011, Kaufmann et al., 2012, Cappé et al., 2013). Recently, Kaufmann and Kalyanakrishnan (2013) proposed KL-LUCB, an extension of LUCB (Kalyanakrishnan et al., 2012), which takes into account the empirical KL divergence between the arms in the case of Bernoulli distributions. More precisely, they considered the Chernoff information, $d^*(\nu_1,\nu_2)$, a symmetric version of the KL divergence for two Bernoulli distributions ν_1 and ν_2, that is defined as

$$d^*(\nu_1,\nu_2) = KL(\nu*,\nu_1) = KL(\nu*,\nu_2), \tag{5.26}$$

where ν^* is the unique Bernoulli distribution such that $KL(\nu^*,\nu_1) = KL(\nu^*,\nu_2)$.

4.2.4 Adaptive Algorithms in the Fixed Budget Setting

In the fixed budget setting, a drawback of GapE, GapE-V, UGapEb, and UGapEb-V is that the exploration parameter a should be tuned according to the complexities H_ϵ and H_ϵ^σ of the multi-bandit problem, which are rarely known in advance. Let us consider for simplicity the case $\epsilon = 0$ and drop the dependency in ϵ. A straightforward solution to this issue is to move to an adaptive version of these algorithms by substituting H and H^σ with suitable estimates $\widehat{H}$ and $\widehat{H}^\sigma$. At each step t of the adaptive (U)GapE and (U)GapE-V algorithms, we estimate these complexities as

$$\widehat{H}(t) = \sum_{p,k} \frac{b^2}{\mathrm{UCB}_{\Delta_{pk}}(t)^2},$$

$$\widehat{H}^{\sigma}(t) = \sum_{p,k} \frac{\Big(\mathrm{LCB}_{\sigma_{pk}}(t) + \sqrt{\mathrm{LCB}_{\sigma_{pk}}(t)^2 + (16/3)b \times \mathrm{UCB}_{\Delta_{pk}}(t)}\Big)^2}{\mathrm{UCB}_{\Delta_{pk}}(t)^2},$$

$$\text{where} \quad \mathrm{UCB}_{\Delta_{pk}}(t) = \widehat{\Delta}_{pk}(t-1) + \sqrt{\frac{1}{2T_{pk}(t-1)}}$$

$$\text{and} \quad \mathrm{LCB}_{\sigma_{pk}}(t) = \max\left(0, \widehat{\sigma}_{pk}(t-1) - \sqrt{\frac{2}{T_{pk}(t-1)-1}}\right).$$

Similar to the adaptive version of UCB-E in Audibert et al. (2010), $\widehat{H}$ and $\widehat{H}^{\sigma}$ are lower-confidence bounds on the true complexities H and H^{σ}. Note that the (U)GapE and (U)GapE-V bounds written for the optimal value of a indicate an inverse relation between the complexity and exploration. By using a lower-bound on the true H and H^{σ}, the algorithms tend to explore arms more uniformly and this allows them to increase the accuracy of their estimated complexities. Although we do not analyze these algorithms, we empirically show in Section 5 that they are in fact able to match the performance of the non-adaptive (U)GapE and (U)GapE-V algorithms.

5 Numerical Simulations

In this section, we report numerical simulations of the gap-based algorithms presented in this chapter, (U)GapE and (U)GapE-V, and their adaptive versions A-(U)GapE and A-(U)GapE-V. The first set of experiments only involves GapE and its variants. We display the advantages of the gap-based algorithms over different baseline methods and discuss their performance on clinical data and in the CBPI rollout allocation problem. We then report the results of the generic algorithm UGapE in both fixed confidence and fixed budget settings. In the fixed budget setting, we highlight that UGapEb and GapE have similar performance, and in the fixed confidence setting, we show that UGapEc outperforms the existing baseline algorithms.

5.1 Results for GapE and its Variants

In this section, we report numerical simulations of GapE and GapE-V, and their adaptive versions A-GapE and A-GapE-V. First we compare them using synthetic problems with *Unif* and *Unif+UCB-E* algorithms introduced in Section. 4. The results of our experiments indicate that **1)** GapE successfully adapts its allocation strategy to the complexity of each bandit and outperforms the uniform allocation strategies, **2)** the use of the empirical variance in GapE-V can significantly improve the performance over GapE, and **3)** the adaptive versions of GapE and GapE-V that estimate the complexities H and H^{σ} online attain the same performance as the basic algorithms, which receive H and H^σ as an input.[7] We highlight the advantage of the dynamic allocation strategies other the static allocation strategies in practice. Finally, additional experiments are run to discuss the applicability of the GapE algorithms on real-world clinical data and on the CBPI rollout allocation problem.

• **Experimental Setting for the Synthetic Problems:** We use the following seven problems in our experiments. Note that $b = 1$ and that a Rademacher distribution with parameters (x, y) takes value x or y with probability 1/2. We remind the reader that GapE only deals with the special case $m = 1$ and $\epsilon = 0$.

◦ *Problem 1.* $n = 700$, $M = 2$, $K = 4$. The arms have Bernoulli distributions with parameters: *bandit 1*: $(0.5, 0.45, 0.4, 0.3)$, *bandit 2*: $(0.5, 0.3, 0.2, 0.1)$.

◦ *Problem 2.* $n = 1000$, $M = 2$, $K = 4$. The arms have Rademacher distributions with parameters (x, y): *bandit 1*: $\{(0, 1.0), (0.45, 0.45), (0.25, 0.65), (0, 0.9)\}$ and in *bandit 2*: $\{(0.4, 0.6), (0.45, 0.45), (0.35, 0.55), (0.25, 0.65)\}$.

◦ *Problem 3.* $n = 1400$, $M = 4$, $K = 4$. The arms have Rademacher distributions with parameters (x, y): *bandit 1*: $\{(0.4, 0.85), (0.25, 0.9), (0.2, 0.95), (0.1, 1.0)\}$, *bandit 2*: $\{(0.0, 1.0), (0.0, 0.8), (0.0, 0.5), (0.3, 0.4)\}$, *bandit 3*: $\{(0.4, 1.0), (0.0, 0.5), (0.1, 0.5), (0.2, 0.5)\}$, and *bandit 4*: $\{(0.0, 1.0), (0.0, 0.8), (0.45, 0.45), (0.45, 0.45)\}$.

◦ *Problem 4.* $n = 400$, $M = 4$, $K = 4$. The arms have Rademacher distributions with the following parameters (x, y): *bandit 1*: $\Big\{(0.15, 0.55), (0.25, 0.5), (0.15, 0.2), (0.75, 0.8)\Big\}$, *bandit 2*: $\Big\{(0.25, 0.45), (0.45, 0.85), (0.2, 0.8), (0.2, 0.8)\Big\}$, *bandit 3*: $\Big\{(0.5, 1.0), (0.6, 0.75), (0.5, 0.6), (0.2, 0.4)\Big\}$, *bandit 4*: $\Big\{(0, 0.9), (0, 0.5), (0.5, 0.5), (0.3, 0.85)\Big\}$.

◦ *Problem 5.* $n = 700$, $M = 3$, $K = 3$. The arms have Rademacher distributions with the following parameters (x, y): *bandit 1*: $\Big\{(0.65, 1.0), (0.35, 0.95), (0.15, 0.6)\Big\}$, *bandit 2*: $\Big\{(0.3, 0.5), (0.5, 0.6), (0.3, 0.6)\Big\}$, *bandit 3*: $\Big\{(0.0, 0.45), (0.3, 0.9), (0.55, 0.6)\Big\}$.

◦ *Problem 6.* $n = 1500$, $M = 10$, $K = 4$. The arms have Rademacher distributions with the following parameters (x, y): *bandit 1*: $\Big\{(0.9, 0.9), (0.5, 0.7), (0, 0.55), (0.15, 0.25)\Big\}$, *bandit 2*: $\Big\{(0.15, 0.60), (0.35, 0.75), (0.4, 0.85), (0.15, 0.65)\Big\}$, *bandit 3*: $\Big\{(0.4, 0.55), (0.05, 0.85), (0, 0.45), (0.2, 0.25)\Big\}$, *bandit 4*: $\Big\{(0.85, 1.0), (0.15, 0.35), (0.2, 0.4), (0.15, 0.9)\Big\}$, *bandit 5*: $\Big\{(0.25, 0.75), (0.15, 0.75), (0.9, 0.95), (0.4, 0.95)\Big\}$, *bandit 6*: $\Big\{(0.45, 0.65), (0.85, 1.0), (0.4, 0.8), (0.2, 0.9)\Big\}$, *bandit 7*: $\Big\{(0, 0.85), (0.3, 0.5), (0.4, 1.0),$

[7]We drop the dependency in ϵ in the complexity terms when discussing the GapE results.

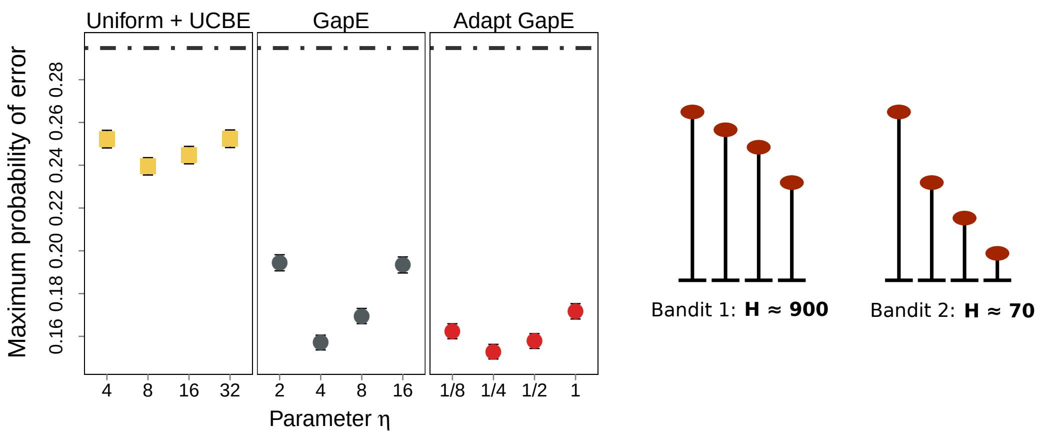


Figure 5.7: Problem 1: Comparison between GapE, adaptive GapE, and the uniform strategies. Maximum probability of error of the algorithms *(left)* and the graphical illustration of the problem *(right)*.

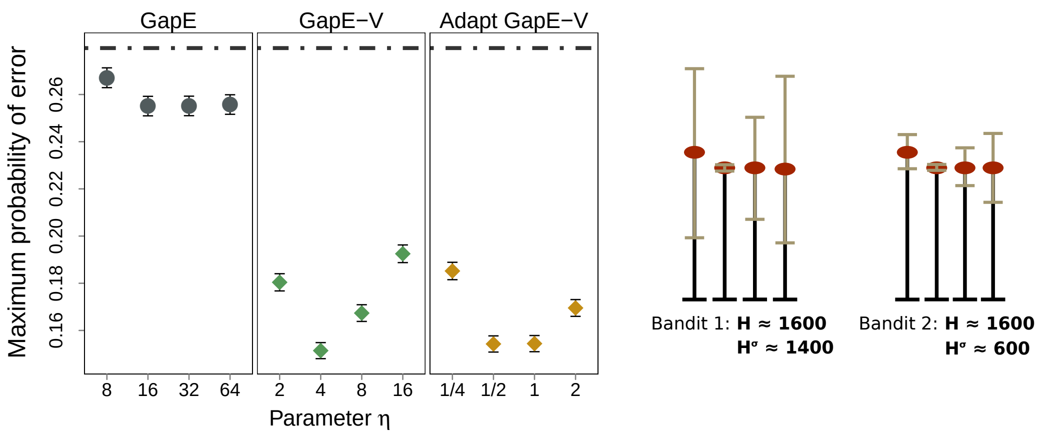


Figure 5.8: Problem 2: Comparison between GapE, GapE-V, and adaptive GapE-V algorithms. Maximum probability of error of the algorithms *(left)* and the graphical illustration of the problem *(right)*.

$(0.35, 0.4)\big\}$, *bandit 8*: $\big\{(0.55, 0.85), (0.35, 0.75), (0.35, 0.5), (0.25, 1.0)\big\}$, *bandit 9*: $\big\{(0.4, 0.6), (0.55, 0.95), (0.15, 0.6), (0.1, 0.8)\big\}$, *bandit 10*: $\big\{(0.05, 0.3), (0.8, 0.85), (0.2, 0.75), (0.2, 0.75)\big\}$.

∘ *Problem 7.* $n = 3000$, $M = 4$, $K = 4$. The four bandits are identical and the arms have Bernoulli distributions with the following means: $(0.5, 0.45, 0.4, 0.3)$.

• **Experimental Results for the Synthetic Problems:** All the algorithms, except the uniform allocation, have an exploration parameter a. The theoretical analysis suggests that a should be proportional to $\frac{n}{H}$. Although a could be optimized according to the bound, since the constants in the analysis are not accurate, we will run the algorithms with $a = \eta \frac{n}{H}$, where η is a parameter which is empirically tuned (in the experiments we report four different values of η). If H correctly defines the complexity of the exploration problem (i.e., the number of samples to find the best arms with high probability), η should simply correct the inaccuracy of the constants in the analysis, and thus, the range of its nearly-optimal values should be constant across different problems. In *Unif+UCB-E*, UCB-E is run with the budget of n/M and the same parameter η for all the bandits. Finally, we set $n \simeq H^\sigma$, since we expect H^σ to roughly capture the number of pulls necessary to solve the pure exploration problem with high probability. In the following figures, we report the performance $l(n)$, i.e., the probability to identify the best arm in all the bandits after n rounds, of the gap-based algorithms as well as *Unif* and *Unif+UCB-E* strategies. The results are averaged over 10^5 runs and the error bars correspond to three times the estimated standard deviation. In all the figures the performance of *Unif* is reported as a horizontal dashed line.

The left panel of Figure 5.7 displays the performance of *Unif+UCB-E*, GapE, and A-GapE in Problem 1. As expected, *Unif+UCB-E* has a better performance (23.9% probability of error) than *Unif* (29.4% probability of error), since it adapts the allocation within each bandit so as to pull more often the nearly-optimal arms. However, the two bandit problems are not equally difficult. In fact, their complexities are very different ($H_1 \simeq 925$ and $H_2 \simeq 67$), and thus, much less samples are needed to identify the best arm in the second bandit than in the first one. Unlike *Unif+UCB-E*, GapE adapts its allocation strategy to the complexities of the bandits (on average only 19% of the pulls are allocated to the second bandit), and at the same time to the arm complexities within each bandit (in the first bandit the averaged allocation of GapE is $(37\%, 36\%, 20\%, 7\%)$). As a result, GapE has a probability of error of 15.7%, which represents a significant improvement over *Unif+UCB-E*.

The left panel of Figure 5.8 compares the performance of GapE, GapE-V, and A-GapE-V in Problem 2. In this problem, all the gaps are equal ($\Delta_{pk} = 0.05$), and thus, all the arms (and bandits) have the same complexity $H_{pk} = 400$. As a result, GapE tends to implement a nearly uniform allocation, which results in a small difference between *Unif* and GapE (28% and 25% accuracy, respectively). The reason why GapE is still able to improve over *Unif* may be explained by the difference between static and dynamic allocation strategies and it is further investigated in the upcoming “Twin Bandits” paragraph. Unlike the gaps, the variance of the arms is extremely heterogeneous. In fact, the variance of the arms of bandit 1 is bigger than in bandit 2, thus making it harder to solve. This difference is captured by the definition of H^σ ($H_1^\sigma \simeq 1400 > H_2^\sigma \simeq 600$). Note also that $H^\sigma \leq H$. As discussed in Section 4.2.2, since GapE-V takes into account the empirical variance of the arms, it is able to adapt to the complexity H_{pk}^σ of each bandit-arm pair and to focus more on uncertain arms. GapE-V improves the final accuracy by almost 10% with respect to GapE.

From both Figures 5.7 and 5.8, we also notice that the adaptive algorithms achieve similar performance to their non-adaptive counterparts. Finally, we notice that a good choice of parameter η for GapE-V is always close to 2 and 4 (see also the upcoming additional experiments), while GapE needs η to be tuned more carefully, particularly

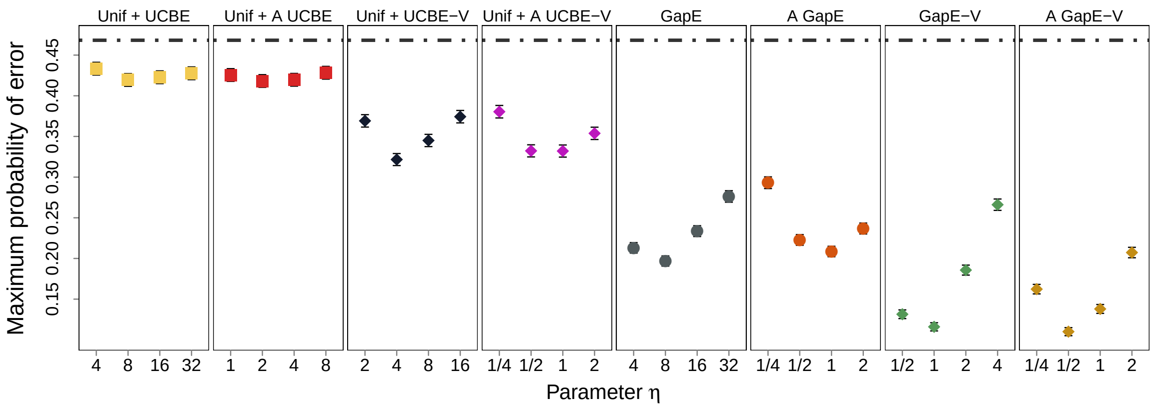


Figure 5.9: Performance of the algorithms in Problem 3.

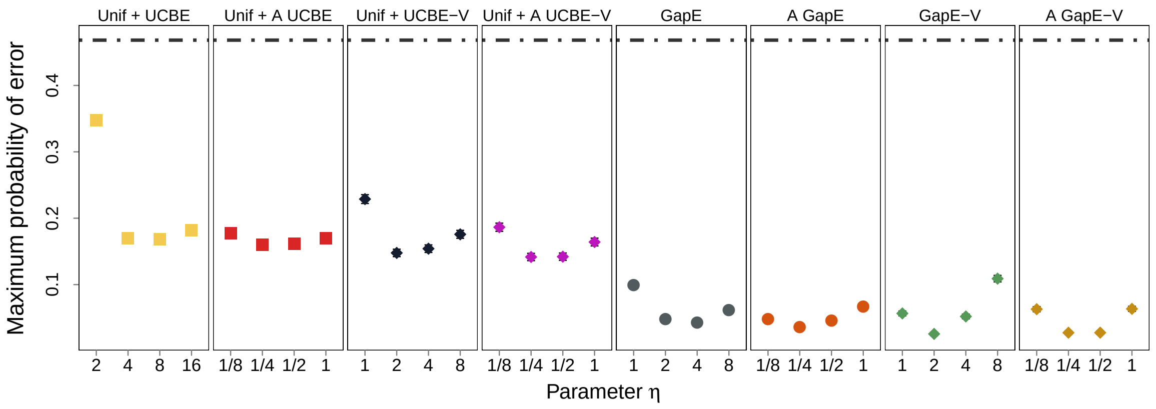


Figure 5.10: Performances of all the algorithms in Problem 4.

in Problem 2 where the large values of η try to compensate the fact that H does not successfully capture the real complexity of the problem. This further strengthens the intuition that H^σ is a more accurate measure of the complexity for the multi-bandit pure exploration problem.

While Problems 1 and 2 are relatively simple, we report the results of the more complicated Problems 3, 4, 5, and 6 in Figures 5.9, 5.10, 5.11, and 5.12, respectively. The experiments are designed so that the complexity with respect to the variance of each bandit and within each bandit is strongly heterogeneous. In fact, in these problems, we randomly generated the parameters x and y of the Rademacher distributions. In order to test the robustness of the algorithms we design problems where the number of arms varies from 9 to 40. In these experiment, we also introduce UCBE-V that extends UCB-E by taking into account the empirical variance similarly to GapE-V. The results mostly confirm those reported before. In fact, in all these problems, all the gap-based algorithms outperform the Unif+UCB-E algorithm. Furthermore, it can be noticed that taking into account the variance leads to an extra improvement of the performance. In these experiments, we again notice that GapE-V has its best performance when the exploration parameter η is in the interval $[2-4]$. This strengthens

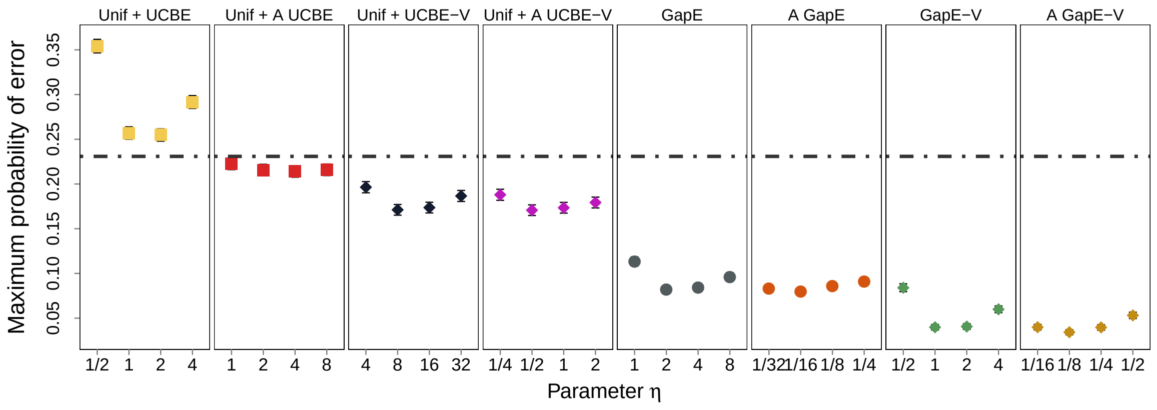


Figure 5.11: Performances of all the algorithms in Problem 5.

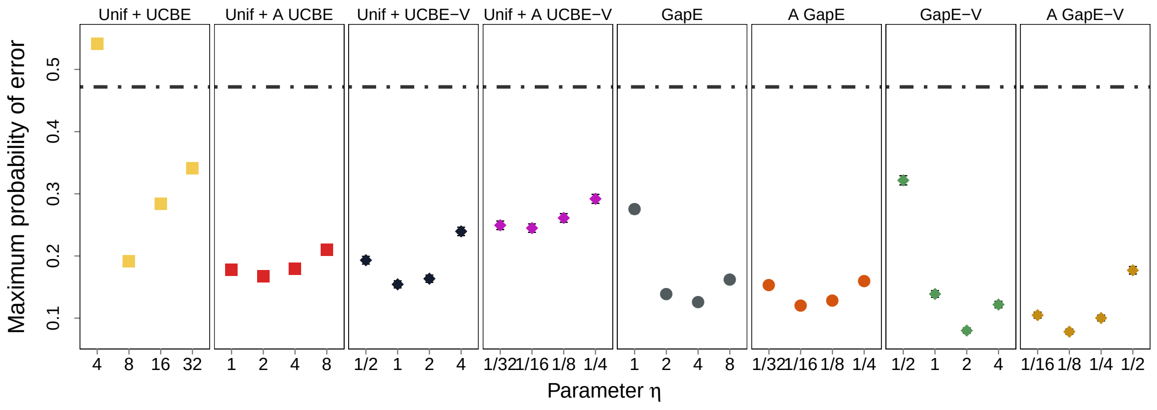


Figure 5.12: Performances of all the algorithms in Problem 6.

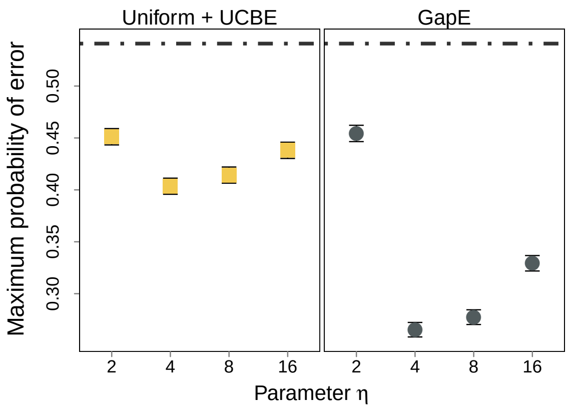


Figure 5.13: Problem 7: The benefit of adaptive allocation over the bandits in the twin bandits problem.

the claim that the complexity H^σ is a good measure of the complexity for any given problem. Moreover, this makes the algorithms easy to use as it provides an easy way to tune the exploration parameter η .

One remark about Problem 6 (see Figure 5.11) is that in this problem we notice that Unif+UCB-E performs worse than Uniform. In bandit 3, the gap between arm 2 and arm 3 is very small (= 0.025). Therefore, the complexity H of this bandit is high, $H_3 \simeq 3000$. However the variance of arm 3 in bandit 3 is really small, thus making H not representative of the true hardness of this problem. The budget n in this experiment is set to 700, and as a result, the budget allocated to the bandit 3 in Unif+UCBE is 233. This budget is small with respect to the complexity H, and thus, the exploration term of UCB-E will be small and almost no exploration will be done in this bandit. This leads Unif+UCB-E to perform worse than Unif. Note that when the exploration parameter η tends to infinity, UCB-E becomes equivalent to the Uniform algorithm. Hence, one can still recover the performance of the Uniform algorithm by setting $\eta \gg 1$.

• **The Twin Bandits (static vs dynamic allocations):** In this paragraph, we consider the Problem 7. Since in this problem the bandits are identical, it seems intuitive to uniformly allocate the budget over the bandits. Hence, we would expect GapE and Unif+UCB-E to have the same performance in this problem. However, Figure 5.13 shows that GapE performs significantly better than Unif+UCB-E. This suggests that dynamic allocation strategies (GapE) might outperform static allocation strategies (Unif+UCB-E). A possible explanation would be that GapE is able to adapt to the *actual* observations. For example, it could happen in one bandit that the empirical mean of the best arm is bigger than its true value, while the empirical means of the sub-optimal arms are smaller than their true values. This means that for this specific realization, the estimated potential simple regret in this bandit is small and the bandit is considered to be easier than what it is. The opposite may happen in another bandit, thus, making it harder than what it is. In this case, more pulls should be allocated to the second bandit because this is where an error of identification is the most likely to happen in this particular realization. Since GapE adapts, in most of the cases, to the observations of each realization of the problem, it seems to successfully adapt to the specific "empirical" hardness of the bandits and to obtain a better performance than an allocation that statically chooses the number of pulls on the basis of the gaps. This result shows a potential advantage of dynamic strategies with respect to static strategies and it asks for a more thorough investigation. Moreover, this phenomenon is also probably related (among others) to the observation that, in practice, adaptive sampling (index-based) algorithms are more efficient than uniform sampling (with rejection/acceptance techniques) algorithms, as briefly mentioned in the introduction of this chapter.

• **Clinical Data:** In this paragraph we compare the performance of GapE to MinMaxPics, one of the algorithms designed by Deng et al. (2011) for the multi-bandit best arm identification problem. MinMaxPics computes at time t an empirical estimation of a "good" static allocation based on the samples collected until time t. This computation is similar the one reported in Remark 5.3. Then MinMaxPics tries to mimic this empirical static allocation with probability $1-\epsilon$ or explore at random with probability ϵ.

Among the reported experiments of their paper, one is based on real world medical data (extracted from Keller et al. (2000)). There, the goal is to find, for three different groups of people (subpopulations with different history of alcoholism), the best of two treatments to cure their depression. The problem is therefore a problem with 3 bandits composed each of two arms.

We report the performances of GapE, MinMaxPics with $\epsilon \in \{.1, .2, .5\}$, the uniform strategy and the population sampling strategy in Figure 5.14. Population sampling correspond to the case where no active selection of patient can be accomplished and those are sampled according to their probability of appearance at the hospital. Figure 5.14 displays the performance in terms of the maximum probability of error with respect to the number of patient tested. The main observation is that GapE outperforms MinMaxPics by a large margin. As the variances in this experiment were not very heterogeneous, GapE-V was not implemented.

• **Application in Rollout Allocation in Classification-based Policy Iteration Algorithms:** Our first objective when designing the gap-based algorithms for the best arm identification problem was to apply them to the problem of rollout allocation in classification-based policy iteration (CBPI) algorithms in reinforcement (see Chapters 1, 3, and 4). In this problem, a set of N' states, called the rollout set $\mathcal{D} = \{s^{(i)}\}_{i=1}^{N'}$, a finite set of actions $\mathcal{A}$, and a current policy π are given. The goal is to compute the greedy policy with respect to policy π. For each state $s^{(i)} \in \mathcal{D}$ and action $a \in \mathcal{A}$, a rollout can be performed whose return is an estimate, $\widehat{Q}(s^{(i)}, a)$, of the action-value function $Q(s^{(i)}, a)$. A policy is then computed as the one in the policy space Π with

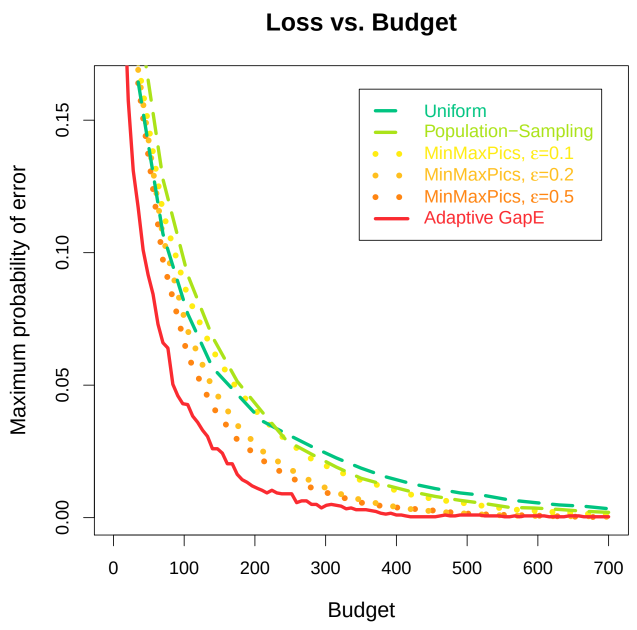


Figure 5.14: Performances of GapE and MinMaxPics in the medical experiment.

minimal cost-sensitive error $\widehat{\mathcal{L}}^{\Pi}(\pi')$ defined as

$$\widehat{\mathcal{L}}^{\Pi}(\pi') = \frac{1}{N'} \sum_{i=1}^{N'} \Big[\max_{a\in\mathcal{A}} \widehat{Q}(s^{(i)}, a) - \widehat{Q}\Big(s^{(i)}, \pi'(s^{(i)})\Big) \Big]. \tag{5.27}$$

The minimization of Equation 5.27 urges to compute a policy π' that recommends at each state $s^{(i)}$ of the rollout set, an action whose empirical action-value $\widehat{Q}\Big(s^{(i)}, \pi'(s^{(i)})\Big)$ is not far from the action with the highest empirical action-value at $s^{(i)}$, i.e., $\max_{a\in\mathcal{A}} \widehat{Q}(s^{(i)}, a)$. The standard procedure up until now in the literature has been to uniformly allocate the rollouts over the states in $\mathcal{D}$ and the actions in $\mathcal{A}$. It is however sub-optimal to allocate the same number of rollouts to the actions with different action-values. One may think that it would be more efficient, once the actions clearly sub-optimal have been discovered, to focus the effort to discriminate between the actions that are likely to be the best. This leads to a best arm identification problem formulation. Moreover, another intuition is that if discriminating the best action in one state is easier than in another, more effort should be allocated to the latter state.

This is why we reformulated the problem of finding the best rollout allocation in order to find a policy close to the greedy policy in Equation 5.27 into the surrogate problem where the objective is to identify the greedy action (*arm*) in each of the states (*bandit*) in a training set. This is the formulation which led us to the design of GapE, and later UGapE. The results of our experiments, partially reported in Gabillon et al. (2010), are somehow deceptive. They first confirm the intuition that a non-uniform allocation of the rollouts among the actions indeed permits to improve the overall performance of the rollout classification-based RL algorithm. On the contrary, when trying to improve the uniform allocation over states, all our efforts with either GapE or its variants failed to bring a significant improvement. The main reason why is not clear to us. Maybe the considered domains of RL (Mountain car and Inverted pendulum) are such that they flatten the benefit of our new approaches. Another important issue may be our formulation of the problem in term of a multi-bandit problem with the objective of minimizing the probability of error while the initial objective was to minimize the cost sensitive loss of the classifier. In this formulation, we argued that one should focus on the states where the best action is the hardest to discriminate (where actions have similar action-values), but one can argue that, in fact those states are not that critical in the CBPI application because, by definition, a mistake there would result in a very small loss in term of action-value. We believe that this phenomenon shadows the interest of sampling non uniformly in the CBPI framework. However, it may be of interest to use an allocation strategy whose target is to minimize, in each bandit, the simple regret rather than the probability of error. Indeed, minimizing the sum of the simple regret over the bandits (states) would correspond exactly with the objective of minimizing the cost-sensitive loss. Designing specific allocation strategies that target the simple regret minimization is nonetheless still an open problem and is left as an interesting future work. More generally, this problem may need another formulation. An interesting direction of research would be to build on or to study the applicability of recent active learning algorithms proposed for cost-sensitive multi-label prediction (Agarwal, 2013) as it would match the use of cost-sensitive classification in CBPI (see Chapter 3 and Chapter 4).

Finally, this rollout allocation problem has connections with the problem of finding a distribution generating the rollout states that improves the efficiency of the CBPI algorithm (this problem can in fact also be encountered in a large number of batch RL algorithms). Indeed, if there was a good heuristic for defining such a distribution, it could also be used to help the rollout allocation focus more on the more "important" states. However, this problem is known to be a hard problem and is still open. This maybe gives another possible explanation for the lack of positive results in our rollout allocation problem.

5.2 Results for UGapE and its Variants

In this section, we compare UGapE and UGapE-V to the state-of-the-art algorithms in the fixed budget and fixed confidence settings in the single-bandit $M = 1$ scenario. The objectives are to show that **1)** UGapEb has the same performance as GapE in practice (in the fixed budget setting) and **2)** the expected improvement of UGapEc-V other the Bernstein Races (see Section 4.2.2) are noticeable in practice (in the fixed confidence setting).

5.2.1 Fixed Confidence Setting

Experimental setting. We define the following two problems, where (x, y) represents a uniform distribution in $[x, y]$:

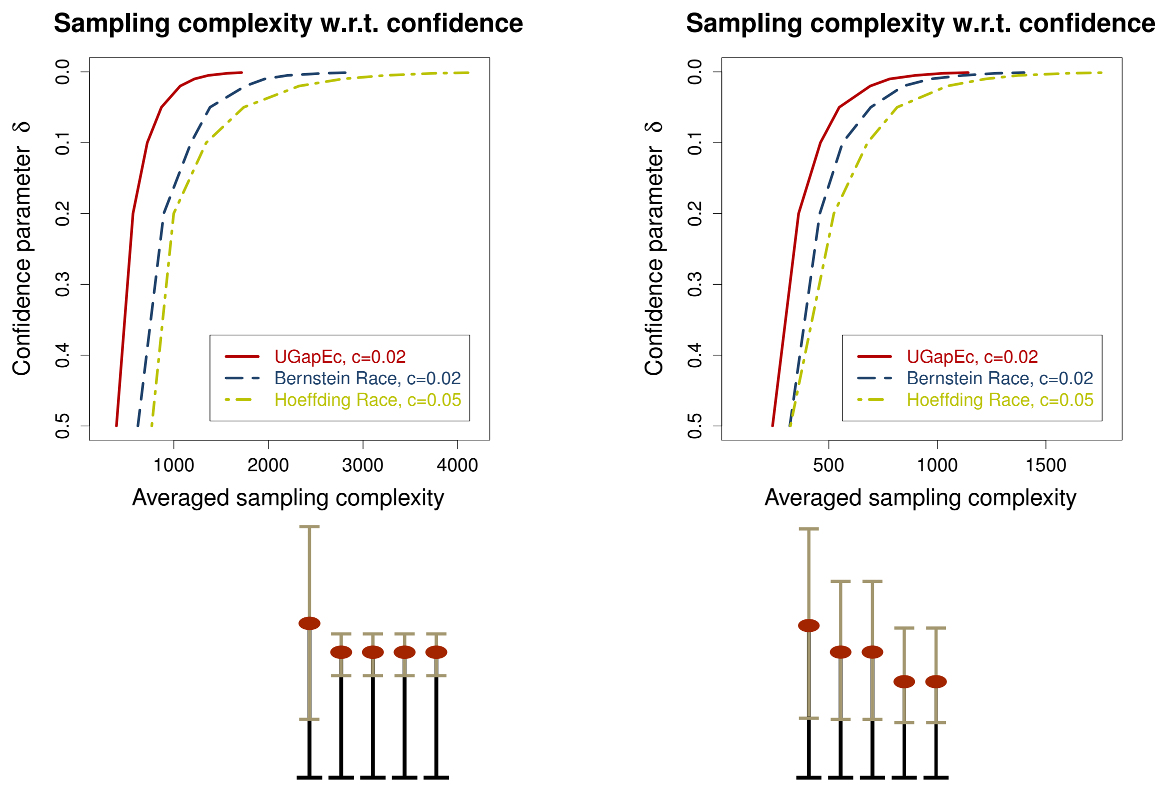
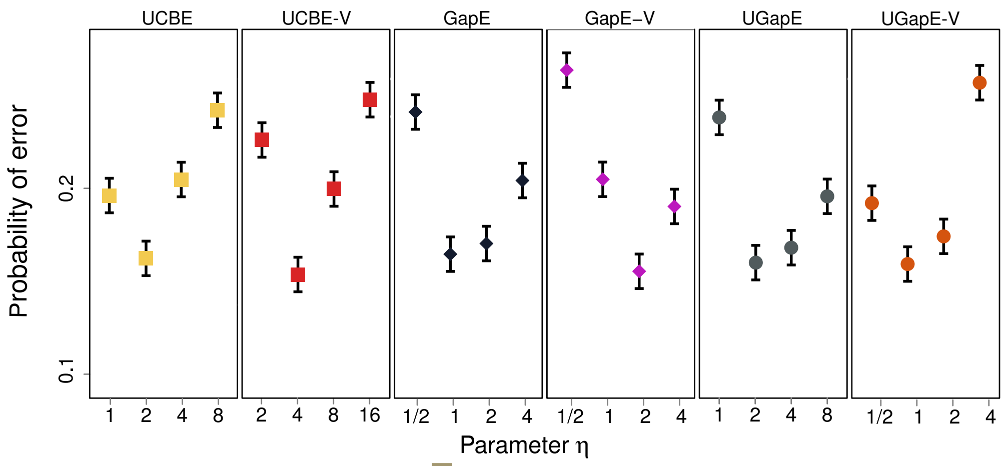


Figure 5.15: Comparison between UGapEc-V, Bernstein Races, and Hoeffding Races algorithms on Problem 8 *(left)* and Problem 9 *(right)*.

- *Problem 8.* $K = 5$ arms with parameters $\big((0,1),(0.4,0.5),(0.4,0.5),(0.4,0.5),(0.4,0.5)\big)$.
- *Problem 9.* $K = 5$ arms with parameters $\big((0,1),(0,0.8),(0,0.8),(0,0.6),(0,0.6)\big)$.

We compare UGapEc-V with the Bernstein and Hoeffding Races algorithms. All the algorithms have an exploration parameter c that we empirically tune. For each algorithm and each confidence parameter δ, we compute the average sample complexity over 1000 runs for different values of $c \in \{1, 0.5, 0.2, 0.1, 0.05, 0.02, 0.01, 0.005\}$. For each algorithm, we only consider the results corresponding to the value of c for which the required confidence level δ is satisfied, i.e., the values of c for which the algorithm satisfies $\mathbb{P}[r_{\Omega(\widetilde{n})} > \epsilon = 0] \leq \delta$, for all the values of δ considered in the experiment. Finally in Figure 5.15, for each algorithm, we report the results for the value of c with the smallest sample complexity.

In the left panel of Figure 5.15, we report the results for Problem 8 with $m = 1$ and $\epsilon = 0$. In this problem, the optimal arm has significantly higher variance than the other arms. This problem has been designed to highlight the difference between the three algorithms and to illustrate the advantage of UGapEc-V over the Racing



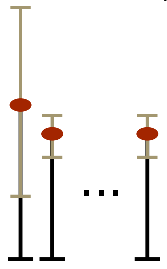
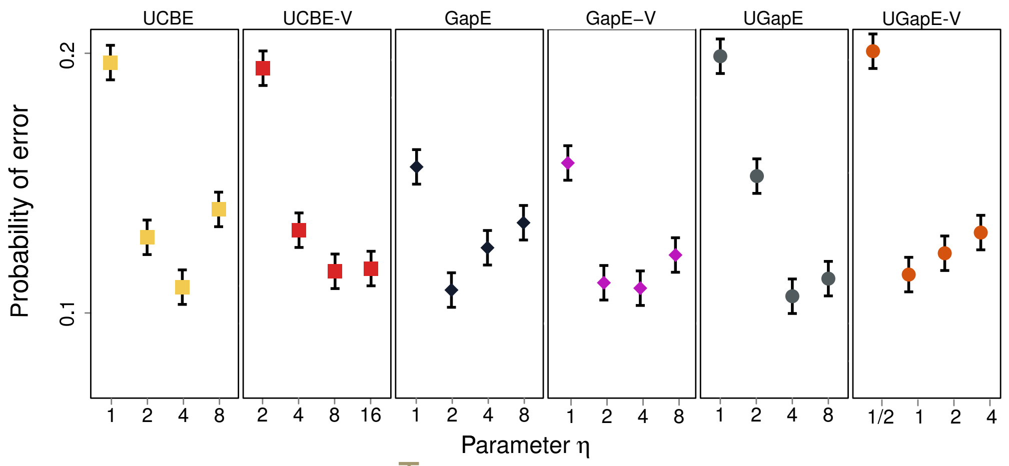


Figure 5.16: Comparison between UCB-E, UCBE-V, GapE, GapE-V, UGapE, and UGapE-V algorithms in Problem 10.

algorithms, as discussed in Section 4.2.2. Since the sub-optimal arms have the same mean and variance, the analysis of Bernstein and Hoeffding Races suggests that all the arms, including the best one, should be pulled the same number of times. However, since the Bernstein Races takes into account the variance, it has a tighter bound and the stopping condition is met earlier than in Hoeffding Races. For example, for $\delta = 0.1$, Bernstein Races has an expected sample complexity of 1181 pulls, while the Hoeffding Races stops after 1342 pulls on average. On the other hand, as expected from the theoretical analysis (see Section 4.2.2), UGapEc-V stops after only 719 pulls. Note that UGapEc-V distributes the number of pulls on average as (72%,7%,7%,7%,7%) over the arms. This indicates that the algorithm successfully adapts to the variance of the arms. The parameter c for which the algorithms have a minimal sample complexity are $c = 0.02$ for Bernstein Races and UGapEc-V and $c = 0.05$ for Hoeffding Races. Finally, in the right panel of Figure 5.15, we show the results for Problem 9. Although unlike Problem 8, this problem has not been specifically designed to illustrate the advantage of UGapEc over the Racing algorithms, UGapEc-V still outperforms the Racing algorithms.

5.2.2 Fixed Budget Setting

In this section, we compare UGapEb with the state-of-the-art fixed budget algorithms: UCBE, UCBE-V, GapE, and GapE-V in the single-bandit scenario $M = 1$. Since all these algorithms share a very similar structure, we expect them to have similar



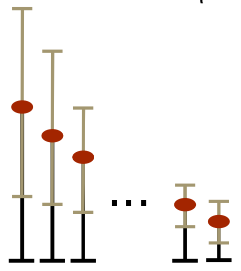


Figure 5.17: Comparison between UCB-E, UCBE-V, GapE, GapE-V, UGapEb, and UGapEb-V algorithms in Problem 11.

performance. All the algorithms have an exploration parameter a. The theoretical analysis suggests that a should be proportional to $\frac{n}{H}$. Although a could be optimized according to the bound, since the constants in the analysis are not accurate, we will run the algorithms with $a = \eta\frac{n}{H}$, where η is a parameter that is empirically tuned (in the experiments we use four different values of η). The results are averaged over 1000 runs and the error bars correspond to three times the estimated standard deviation.

Experimental setting. We use the following two problems in our experiments:

∘ *Problem 10.* $n = 2000$, $K = 20$. The arms have Bernoulli distribution, the best arm has a mean of $1/2$ and the sub-optimal arms have a mean of 0.4.

∘ *Problem 11.* $n = 4000$, $K = 15$. The arms have Bernoulli distribution with parameters $\mu_i = 0.5 - 0.025(i-1)$, $i \in \{1, \dots, 15\}$.

Note that $b = 1$ in these problems. In Figures 5.16 and 5.17, we report the performance, calculated as the probability to identify the best arm after n rounds, of UCB-E, UCBE-V, GapE, GapE-V, UGapEb, and UGapEb-V algorithms. The results indicate that the best performance of each algorithm is achieved for similar values of the parameter η. As expected, all the algorithms achieve similar performance, no one has clear advantage over the others. Investigating the allocation over the budget n

over arms, we also notice that for all the algorithms the number of pulls is inversely proportional to the gaps. We also expect that UGapEb perform very similarly to GapE in the multi-bandit scenario.

6 Summary and Discussion

In this chapter, we studied the problem of multi-bandit best arm identification. We introduced two gap-based exploration algorithms, called GapE and UGapE, and for each algorithm, proved an upper-bound on its probability of error. These are the first theoretical results reported in the multi-bandit setting. We also extended our proposed algorithms to take into account the variance of the arms, and proved upper-bounds for the probability of error of the resulting algorithms. Moreover, we introduced adaptive versions of these algorithms that estimate the complexity of the problem online. The numerical simulations in synthetic problems and in a clinical problem confirmed the theoretical findings that (U)GapE and (U)GapE-V outperform other allocation strategies, and that their adaptive counterparts are able to estimate the complexity without worsening the global performance.

Concerning the application to the rollout allocation problem in the CBPI algorithms, the results are not satisfactory and required further investigation. On one hand, this particular problem seems to be linked to the hard problem of designing a “good” rollout set distribution. On the other hand, to better address the rollout allocation problem, it might be of interest to to design algorithms that minimize the simple regret and not the probability of error.

Although (U)GapE does not know the gaps, experimental results indicate that it can outperform the static allocation strategy, which knows the gaps in advance. This suggests that an adaptive strategy could perform better than a static one. This observation asks for further investigation. Although this investigation may not lead to major changes in the theoretical bounds, understanding this phenomenon could help us to design more efficient algorithms.

We also outlined the link between fixed confidence and fixed budget settings when the complexity of the problem is known to the forecaster. Recovering the same results when ignoring the complexity is an interesting open problem.

Chapter 6
Conclusions and Future Work

This chapter provides a summary of methods and algorithms presented in this thesis, and highlights some open questions that lay ground for future work.

1 Summary

This thesis has been motivated by the study of a relatively novel class of reinforcement learning (RL) algorithms called Classification-based Policy Iteration (CBPI). CBPI algorithms were introduced as a new class of RL algorithms that, contrary to the standard RL algorithms, do not use an explicit representation for value function approximation. Instead, value functions are estimated using rollouts in order to build a training set that consists of states paired with their recommended actions. Using this training set, the greedy policy is then learned as the output of a classifier. Therefore, the policies are no longer defined by their value functions but by the classifier. This observation raises the expectation that CBPI algorithms would perform better than their value function-based counterparts in problems where the policies are simpler to represent than their associated value functions. This thesis stands in part as a novel and strong empirical evidence that the CBPI approach can perform well in such challenging domains such as in the game of Tetris, where standard RL and approximate dynamic programming (ADP) techniques perform poorly. Moreover, we proposed new CBPI methods with better performance than the original algorithms in this class, especially in terms of the number samples (number of interactions with the environment or number of samples generated by the generative model of the system). Our new CBPI algorithms are based on two ideas: **1)** improving the quality of the rollout estimates with the help of a value function approximator (in addition to the rollout estimates) and **2)** sampling adaptively (rather than uniformly) the rollouts over the state-action pairs, which in turn leads to more accurate rollout estimates at state-action pairs that are more crucial to learn the greedy policy. Finally, we proposed a formulation of the CBPI algorithms that allows us to understand them as part of the broader class of ADP methods, namely Approximate Modified Policy Iteration (AMPI) algorithms.

In Chapter 3, we presented our first extension of the standard CBPI algorithms. The idea is to estimate the action-value functions as a combination of a truncated rollout and a value function approximator, called the *critic*, that approximates the value of the state at which the rollout has been truncated. The role of the critic is to reduce the variance of the action-value function estimates at the cost of introducing a bias. We presented a new CBPI algorithm, called *direct policy iteration with a critic* (DPI-Critic), and provided its finite-sample performance analysis when the critic is based on the **1)** least-squares temporal-difference learning (LSTD), and **2)** Bellman residual minimization (BRM) methods. The reported performance bound and our subsequent empirical evaluations in two benchmark RL problems show how our approach can improve the overall performance of the CBPI algorithms, especially when we are given a fixed budget of samples.

In Chapter 4, we studied the approximate version of the modified policy iteration (MPI) algorithm (Puterman and Shin, 1978). We proposed and studied three approximate modified policy iteration (AMPI) algorithms, of which two correspond to the well-known ADP methods: fitted-value iteration and fitted-Q iteration, and one belongs to the class of CBPI algorithms. We derived the first error propagation analysis for the AMPI algorithms that unifies this for approximate policy and value iteration methods. We also provided finite-sample performance bounds for our three AMPI algorithms. Finally, we evaluated the behavior of the proposed algorithms in the mountain car problem as well as in the game of Tetris. Remarkably, we observed that in Tetris, our classification-based MPI (CBMPI) algorithm outperforms existing ADP methods by a large margin, and even obtain better results than the state-of-the-art methods, while using a considerably smaller number of samples.

In Chapter 5, we studied the problem of identifying the best arm(s) in each of the bandits in a multi-bandit multi-armed setting. The initial motivation for this study was the rollout allocation problem in the CBPI algorithms. We showed that this problem can be seen as a multi-bandit problem, for which no theory had been developed. Since the problem of multi-bandit best arm identification has other potential applications in various domains, such as clinical trials and brain computer interface, we studied it as a stand-alone problem. In order to solve the problem, we first proposed two algorithms, called *Gap-based Exploration* (GapE) and *Unified Gap-based Exploration* (UGapE), both focus on the arms with small gap, i.e., arms whose means are close to the mean of the best arm in the same bandit. We then improved upon these algorithms by introducing GapE-V and UGapE-V, which take into account the variance of the arms in addition to their gaps. We proved an upper-bound on the probability of error for all these algorithms. Since GapE and GapE-V need to tune an exploration parameter that depends on the complexity of the problem, which is often unknown a priori, we also introduced variations of these algorithms that estimate this complexity online. Finally, we evaluated the performance of our algorithms and compared them against other allocation strategies in a number of synthetic problems as well as a real-world clinical problem. We also evaluated the resulting algorithms in the rollout allocation problem in the CBPI algorithms. Unfortunately, the CBPI results were relatively disappointing, and thus, the problem requires further investigation.

2 Future Work

This work opens ground for a number of exciting and interesting research directions which are briefly outlined below.

2.1 Implementation of Cost-Sensitive Classifiers for CBPI

The use of a cost-sensitive classifier plays a key role in the CBPI algorithms. However in our experiments, we did not implement it in its original formulation. In the experiments reported in Chapter 3, the loss minimized was the 0-1 loss instead of the cost-sensitive loss. For the experiments in the game of Tetris, we minimized the cost-sensitive loss. This was done by using an optimization technique namely the cross entropy method. Nevertheless, instead of representing and learning the policies as mappings from states to actions, the policies were indirectly represented by evaluation functions which are mappings from state-action pair to a real values (as the value functions are). This use of evaluation functions somehow brings CBPI methods close to regression-based (value-based) methods, though the loss used to learn them in both cases are different. Therefore it would be interesting to implement multi-class cost-sensitive classifiers that directly learn policies rather than evaluation functions. However, the design of multi-class cost-sensitive classifier is a challenging problem for most of the standard and popular classification techniques such as SVM (Masnadi-Shirazi et al., 2012) and Multiboost (Benbouzid et al., 2012). Recently, Pires et al. (2013) have provided some results on multi-class cost-sensitive classification that may be useful in the CBPI algorithms.

2.2 Adaptive Sampling of the Rollout Set and the Rollouts in CBPI

One of the initial objectives of this thesis was to design adaptive allocation strategies for the rollout in the CBPI algorithms. Our idea was to cast this problem as a pure exploration bandit problem and obtain better results, but unfortunately, we were not able to fulfill this objective in this dissertation. One possible reason is that the strategies designed for the bandit problem aim to minimize the probability of the incorrect identification of the best arm (greedy action) in any of the states of the rollout set. Therefore, if in one state of the rollout set, two actions have extremely close action-value functions, the strategies will use many rollouts to discriminate between the two actions in this particular state and deprive the other rollout states. However, this is not the behavior we expect. It would be more natural to try to minimize the simple regret over all the rollout states, because in this case, discriminating between two actions with close action-value functions would not be seen as a priority, as the potential simple regret to incur is small. This CBPI rollout allocation problem shows that minimizing the simple regret can be a different objective from minimizing the probability of error as these two objectives require different sampling approaches. However, to the best of our knowledge, strategies to directly minimize the simple regret have not yet been considered. Instead, minimizing the probability of error has been used as a surrogate **1)** since the probability of error is both an upper bound and a lower bound on the simple regret (up to constants depending on the gaps) and therefore minimizing one permits to guarantee that the other one is relatively small and **2)** because asymptotically (when n, the number of pulls tends to infinity), both minimizing the probability of error or the simple regret are equivalent (the static allocations derived from the Chernoff-Hoeffding bound are the same). These observations call for the investigation of novel strategies that aim directly at minimizing the simple regret.

Finally, this rollout allocation problem has connections with the problem of finding a distribution generating the rollout states that improves the efficiency of the CBPI algorithm (this problem can in fact also be encountered in a large number of batch RL algorithms). Indeed, if there was a good heuristic for defining such a distribution, it could also be used to help the rollout allocation focus more on the more "important" states. However, this problem is known to be difficult and is still open. Perhaps, this serves as yet another possible explanation for the lack of positive results in our rollout allocation problem. An interesting direction of research would be to build upon, or to study the applicability of, recent active learning algorithms proposed for cost-sensitive multi-label prediction (Agarwal, 2013) as it would match the use of cost-sensitive classification in CBPI (see Chapter 3 and Chapter 4). Another interesting reference is the work by Rexakis and Lagoudakis (2012) where a heuristic is proposed in order to sample the states of the rollout set in the CBPI framework. This strategy is based on the use of an SVM classifier.

2.3 Tetris

The performance of CBMPI in the game of Tetris is encouraging. While there still exist some technical issues in the code to be addressed in order to make CBMPI perform even better in Tetris, here, we report some possible algorithmic modifications that may, in addition, lead to improving the performance of CBMPI in Tetris. We also discuss how the comparison with other competing methods should be modified.

On the CBMPI side, a possible approach to improve the performance would be to make the policy updates asynchronous. The goal here would be to speed up the learning process (in terms of the number of samples used) by only renewing a limited portion of the training set at each iteration. One may also consider larger policy spaces with possibly stochastic and/or non-stationary policies in a fashion similar to Scherrer and Lesner (2012).

Another class of policy search methods that have been applied to Tetris are policy gradient algorithms that have not performed as well as CBMPI in this game. A closer look reveals that their poor results are probably due to their use of the features that have not shown good performance in the game of Tetris (the Bertsekas features). In order to have a more thorough comparison, these algorithms, and especially the recent approach of Furmston and Barber (2012), should be tested using the Dellacherie features. Moreover, in order to conduct a fair comparison, we should evaluate the performance of the new variation of the cross entropy algorithms introduced by Goschin et al. (2013) and discussed in Section 5.2.1 of Chapter 4, in the original version of Tetris.

Finally, this work leaves open the question of understanding why value function-based ADP algorithms fail in the game of Tetris. We conjecture that this could be addressed by going beyond the standard linear value function architecture. In particular, one may use the Poisson model for value function, whose variance grows exponentially with the height of the board. On the same issue, the piecewise-linear models depending on the height of the board, or even Gaussian process regression tools may prove useful.

2.4 Improvements in the Best Arm Identification Problem

Our approach to the best arm identification problems have been motivated both by practical efficiency and theoretical guarantees. There is room for improvement in both fronts. On the practical side, looking at the fixed budget setting, GapE has good performance but needs its parameter to be tuned, while Successive Rejects is parameter free with the cost of a slightly diminished performance. It would be of interest to design new efficient strategies for the best arm identification problem that are both efficient and parameter free. An interesting direction of research is to consider approaches inspired by Thompson sampling strategies in the cumulative regret setting. Indeed, these methods have been shown to have strong theoretical guarantees (Kaufmann et al., 2012) and to be extremely efficient in practice (Chapelle and Li, 2011). On the theoretical side, in the fixed budget setting, there still exists a gap between the existing lower and upper bounds in the case where the complexity is unknown to the forecaster. Matching the two would allows us to have a better idea of what the right measure of complexity is for this setting and to see if it corresponds to the same measure of complexity as in the fixed confidence setting. In this case, there would be a strong evidence that a common sampling routine may be designed for both settings that leads to optimal bounds in each case. Also on the theoretical side, it would be of interest to characterize why, as noticed in some of our experiments, dynamic allocations significantly outperform static allocations while our current analysis expect them to have similar performance.

Another direction is to change the objective as motivated in Section 2.2 of this chapter. It would be interesting to design strategies that directly attempt to minimize the simple regret rather than the probability of error. Finally, the best arm identification problem could be extended to combinatorial spaces. An example of such a setting is a stochastic shortest path problem in a graph where the time taken in each edge would be modeled by a distribution. The goal would then be to identify the shortest path using n pulls, where at each pull the forecaster could only collect a sample from one chosen edge (and not the whole path).

Bibliography

Y. Abbasi-Yadkori, D. Pál, and Cs. Szepesvári. Improved Algorithms for Linear Stochastic Bandits. In *Proceedings of the Advances in Neural Information Processing Systems 25*, pages 2312–2320, 2011. (→ page 26.)

A. Agarwal. Selective sampling algorithms for cost-sensitive multiclass prediction. In *Proceedings of the Thirtieth International Conference on Machine Learning*, 2013. (→ pages 135 and 144.)

A. Antos, R. Munos, and Cs. Szepesvári. Fitted Q-iteration in continuous action-space MDPs. In *Proceedings of the Advances in Neural Information Processing Systems 21*, pages 9–16, 2007. (→ pages XIV, 8, 19, 58, and 61.)

A. Antos, Cs. Szepesvári, and R. Munos. Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning Journal*, 71:89–129, 2008. (→ page 37.)

A. Antos, V. Grover, and Cs. Szepesvári. Active Learning in Heteroscedastic Noise. *Theoretical Computer Science*, 411(29-30):2712–2728, 2010. (→ pages 26 and 122.)

J.-Y. Audibert, R. Munos, and Cs. Szepesvári. Tuning Bandit Algorithms in Stochastic Environments. In *Proceedings of the Eighteenth International Conference on Algorithmic Learning Theory*, pages 150–165, 2007. (→ pages 25 and 123.)

J.-Y. Audibert, S. Bubeck, and R. Munos. Best Arm Identification in Multi-Armed Bandits. In *Proceedings of the Twenty-Third Conference on Learning Theory*, pages 41–53, 2010. (→ pages XII, XV, 6, 9, 27, 28, 29, 30, 104, 105, 106, 112, 113, 114, 115, and 126.)

P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multi-armed bandit problem. *Machine Learning*, 47:235–256, 2002. (→ pages 25, 104, 105, and 109.)

P. Auer, N. Cesa-Bianchi, Y. Freund, and R. Schapire. The Nonstochastic Multiarmed Bandit Problem. *SIAM Journal on Computing*, 32(1):48–77, Jan. 2003. (→ page 26.)

L. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 30–37. Morgan Kaufmann, 1995. (→ pages XIII, 8, and 34.)

A. Barto, R. Sutton, and C. Anderson. Neuron-Like Elements that can Solve Difficult Learning Control Problems. *IEEE Transaction on Systems, Man and Cybernetics*, 13:835–846, 1983. (→ pages XI, 6, 23, and 34.)

R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1957. (→ page 17.)

D. Benbouzid, R. Busa-Fekete, N. Casagrande, F.-D. Collin, and B. Kégl. MULTI-BOOST: A Multi-purpose Boosting Package. *Journal Machine Learning Research*, 13:549–553, Mar. 2012. (→ pages 90 and 143.)

D. Bertsekas and S. Ioffe. Temporal Differences-Based Policy Iteration and Applications in Neuro-Dynamic Programming. Technical report, MIT, 1996. (→ pages IX, 3, 17, 19, 58, and 79.)

D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996. (→ pages 15, 16, 19, 58, 65, 66, 70, and 81.)

S. Bradtke and A. Barto. Linear Least-Squares Algorithms for Temporal Difference Learning. *Journal of Machine Learning*, 22:33–57, 1996. (→ pages XIII, 8, and 34.)

S. Bubeck, R. Munos, and G. Stoltz. Pure Exploration in Multi-Armed Bandit Problems. In *Proceedings of the Twentieth International Conference on Algorithmic Learning Theory*, pages 23–37, 2009. (→ pages XII, 6, 27, 28, 104, and 105.)

H. Burgiel. How to Lose at Tetris. *Mathematical Gazette*, 81:194–200, 1997. (→ page 78.)

Z. Cai, D. Zhang, and B. Nebel. Playing tetris using bandit-based Monte-Carlo planning. In *AISB Symposium: AI and Games*, 2011. (→ page 80.)

P. Canbolat and U. Rothblum. (Approximate) iterated successive approximations algorithm for sequential decision processes. *Annals of Operations Research*, pages 1–12, 2012. (→ page 70.)

O. Cappé, A. Garivier, O. Maillard, R. Munos, and G. Stoltz. Kullback-leibler upper confidence bounds for optimal sequential allocation. *Annals of Statistics*, 41(3): 1516–1541, 2013. (→ pages 25 and 125.)

A. Carpentier, A. Lazaric, M. Ghavamzadeh, R. Munos, and P. Auer. Upper-Confidence-Bound Algorithms for Active Learning in Multi-armed Bandits. In *Proceedings of the Twenty-Second International Conference on Algorithmic Learning Theory*, pages 189–203, 2011. (→ page 26.)

C. Chang and C. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27:1–27:27, May 2011. (→ page 75.)

O. Chapelle and L. Li. An empirical evaluation of thompson sampling. In *Proceedings of the Advances in Neural Information Processing Systems 25*, pages 2249–2257, 2011. (→ pages 26 and 145.)

V. Dani, T. Hayes, and S. Kakade. The Price of Bandit Information for Online Optimization. In *Proceedings of the Advances in Neural Information Processing Systems 21*, 2007. (→ page 26.)

E. Demaine, S. Hohenberger, and D. Liben-Nowell. Tetris is Hard, Even to Approximate. In *Proceedings of the Ninth International Computing and Combinatorics Conference*, pages 351–363, 2003. (→ pages IX, 3, and 78.)

K. Deng, J. Pineau, and S. Murphy. Active Learning for Personalizing Treatment. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, 2011. (→ pages 30, 106, 107, and 133.)

C. Dimitrakakis and M. Lagoudakis. Rollout Sampling Approximate Policy Iteration. *Machine Learning Journal*, 72(3):157–171, 2008. (→ page 45.)

D. Ernst, P. Geurts, and L. Wehenkel. Tree-Based Batch Mode Reinforcement Learning. *Journal of Machine Learning Research*, 6:503–556, 2005. (→ pages XIV, 8, 19, 58, and 61.)

E. Even-Dar, S. Mannor, and Y. Mansour. Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems. *Journal of Machine Learning Research*, 7:1079–1105, 2006. (→ pages 27, 28, 105, and 122.)

C. Fahey. Tetris AI, Computer plays Tetris, 2003. `http://colinfahey.com/tetris/tetris.html`. (→ pages 79 and 81.)

A.-M. Farahmand, R. Munos, and Cs. Szepesvári. Error Propagation for Approximate Policy and Value Iteration. In *Proceedings of the Advances in Neural Information Processing Systems 24*, pages 568–576, 2010. (→ pages 68, 70, and 97.)

A.-M. Farahmand, D. Precup, and M. Ghavamzadeh. Generalized Classification-based Approximate Policy Iteration. In *Proceedings of the European Workshop on Reinforcement Learning (EWRL)*, pages 1–11, June 2012. (→ page 44.)

V. Farias and B. Van Roy. *Tetris: A Study of Randomized Constraint Sampling.* Springer-Verlag, 2006. (→ pages IX, 3, and 79.)

A. Fern, S. Yoon, and R. Givan. Approximate policy iteration with a policy language bias. In *Proceedings of the Advances in Neural Information Processing Systems 18*, 2004. (→ pages VIII, X, 2, 4, and 20.)

A. Fern, S. Yoon, and R. Givan. Approximate Policy Iteration with a Policy Language Bias: Solving Relational Markov Decision Processes. *Journal of Artificial Intelligence Research*, 25:75–118, 2006. (→ pages VIII, X, 2, 4, 20, 34, 58, and 61.)

T. Furmston and D. Barber. A Unifying Perspective of Parametric Policy Search Methods for Markov Decision Processes. In *Proceedings of the Advances in Neural Information Processing Systems 26*, pages 2726–2734, 2012. (→ pages 79, 90, and 144.)

V. Gabillon, A. Lazaric, and M. Ghavamzadeh. Rollout Allocation Strategies for Classification-based Policy Iteration. In *Workshop on Reinforcement Learning and Search in Very Large Spaces*, 2010. (→ page 134.)

V. Gabillon, M. Ghavamzadeh, A. Lazaric, and S. Bubeck. Multi-Bandit Best Arm Identification. In *Proceedings of the Advances in Neural Information Processing Systems 25*, pages 2222–2230, 2011a. (→ pages 103, 113, and 125.)

V. Gabillon, A. Lazaric, M. Ghavamzadeh, and B. Scherrer. Classification-based Policy Iteration with a Critic. In *Proceedings of the Twenty-Eighth International Conference on Machine Learning*, pages 1049–1056, 2011b. (→ pages 33 and 58.)

V. Gabillon, M. Ghavamzadeh, and A. Lazaric. Best Arm Identification: A Unified Approach to Fixed Budget and Fixed Confidence. In *Proceedings of the Advances in Neural Information Processing Systems 26*, pages 3221–3229, 2012. (→ page 103.)

V. Gabillon, M. Ghavamzadeh, and B. Scherrer. Approximate Dynamic Programming Finally Performs Well in the Game of Tetris. In *Proceedings of the Advances in Neural Information Processing Systems 27*, 2013. (→ pages 57 and 88.)

M. Ghavamzadeh and A. Lazaric. Conservative and Greedy Approaches to Classification-Based Policy Iteration. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012. (→ page 22.)

S. Goschin, A. Weinstein, and M. Littman. The Cross-Entropy Method Optimizes for Quantiles. In *Proceedings of the Thirtieth International Conference on Machine Learning*, pages 1193–1201, 2013. (→ pages 83 and 144.)

N. Hansen and A. Ostermeier. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation*, 9:159–195, 2001. (→ page 82.)

J. Honda and A. Takemura. An asymptotically optimal policy for finite support models in the multiarmed bandit problem. *Machine Learning*, 85(3):361–391, 2011. (→ page 125.)

R. Howard. *Dynamic Programming and Markov Processes*. The MIT Press, Cambridge, MA, 1960. (→ pages 14 and 16.)

S. Kakade. A natural policy gradient. In *Proceedings of the Advances in Neural Information Processing Systems 15*, pages 1531–1538, 2001. (→ page 79.)

S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the 19th International Conference on Machine Learning*, pages 267–274, 2002. (→ page 22.)

S. Kalyanakrishnan. *Learning Methods for Sequential Decision Making with Imperfect Representations*. PhD thesis, Department of Computer Science, The University of Texas at Austin, Austin, Texas, USA, December 2011. Published as UT Austin Computer Science Technical Report TR-11-41. (→ page 123.)

S. Kalyanakrishnan and P. Stone. Efficient Selection of Multiple Bandit Arms: Theory and Practice. In *Proceedings of the Twenty-Seventh International Conference on Machine Learning*, pages 511–518, 2010. (→ pages 28 and 106.)

S. Kalyanakrishnan, A. Tewari, P. Auer, and P. Stone. PAC Subset Selection in Stochastic Multi-armed Bandits. In *Proceedings of the Twentieth International Conference on Machine Learning*, 2012. (→ pages 28, 112, 117, and 125.)

Z. Karnin, T. Koren, and O. Somekh. Almost Optimal Exploration in Multi-Armed Bandits. In *Proceedings of the Thirtieth International Conference on Machine Learning*, 2013. (→ pages 28, 29, and 123.)

É. Kaufmann and S. Kalyanakrishnan. Information complexity in bandit subset selection. In *Proceedings of the Twenty-Sixth Conference on Learning Theory*, pages 228–251, 2013. (→ pages 28, 30, 106, and 125.)

É. Kaufmann, N. Korda, and R. Munos. Thompson sampling: An asymptotically optimal finite-time analysis. In *Proceedings of the Twenty-Fourth International Conference on Algorithmic Learning Theory*, pages 199–213, 2012. (→ pages 26, 125, and 145.)

M. Kearns, Y. Mansour, and A. Ng. Approximate Planning in Large POMDPs via Reusable Trajectories. In *Proceedings of the Advances in Neural Information Processing Systems 14*, pages 1001–1007. MIT Press, 2000. (→ page 64.)

M. Keller, J. McCullough, D. Klein, B. Arnow, D. Dunner, A. Gelenberg, J. Markowitz, C. Nemeroff, J. Russell, and M. Thase. A comparison of nefazodone, the cognitive behavioral-analysis system of psychotherapy, and their combination for the treatment of chronic depression. *New England Journal of Medicine*, 342(20):1462–1470, 2000. (→ page 133.)

M. Lagoudakis and R. Parr. Least-Squares Policy Iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003a. (→ pages VIII, XIII, 2, 8, 19, 34, 45, 58, and 75.)

M. Lagoudakis and R. Parr. Reinforcement Learning as Classification: Leveraging Modern Classifiers. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 424–431, 2003b. (→ pages VIII, X, 2, 4, 20, 34, 58, and 61.)

A. Lazaric, M. Ghavamzadeh, and R. Munos. Analysis of a Classification-based Policy Iteration Algorithm. In *Proceedings of the Twenty-Seventh International Conference on Machine Learning*, pages 607–614, 2010a. (→ pages VIII, X, XI, XIII, 2, 4, 6, 8, 20, 21, 23, 34, 50, 53, 58, 61, 64, 74, and 100.)

A. Lazaric, M. Ghavamzadeh, and R. Munos. Analysis of a Classification-based Policy Iteration Algorithm. Technical Report 00482065, INRIA, 2010b. (→ pages 34, 41, and 42.)

A. Lazaric, M. Ghavamzadeh, and R. Munos. Finite-Sample Analysis of LSTD. In *Proceedings of the Twenty-Seventh International Conference on Machine Learning*, pages 615–622, 2010c. (→ page 38.)

A. Lazaric, M. Ghavamzadeh, and R. Munos. Finite-Sample Analysis of Least-Squares Policy Iteration. *Journal of Machine Learning Research*, 13:3041–3074, 2012. (→ pages 38, 40, and 56.)

O. Maillard, R. Munos, A. Lazaric, and M. Ghavamzadeh. Finite-Sample Analysis of Bellman Residual Minimization. In *Proceedings of the Second Asian Conference on Machine Learning*, 2010. (→ pages 54 and 56.)

S. Mannor and J. Tsitsiklis. The Sample Complexity of Exploration in the Multi-Armed Bandit Problem. *Journal of Machine Learning Research*, 5:623–648, 2004. (→ page 28.)

S. Mannor, R. Rubinstein, and Y. Gat. The Cross Entropy method for Fast Policy Search. In *In Proceedings of the Twentieth International Conference on Machine Learning*, pages 512–519. Morgan Kaufmann, 2003. (→ page 81.)

P. Marbach and J. Tsitsiklis. A Neuro-Dynamic Programming Approach to Call Admission Control in Integrated Service Networks: The Single Link Case. Technical report, Decision Syst. Rep. LIDS-P-2402, Massachusetts Institute of Technology, 1997. (→ pages VII and 1.)

O. Maron and A. Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In *Proceedings of the Advances in Neural Information Processing Systems 7*, 1993. (→ pages XII, 6, 28, 105, and 106.)

H. Masnadi-Shirazi, N. Vasconcelos, and A. Iranmehr. Cost-Sensitive Support Vector Machines. *CoRR*, abs/1212.0975, 2012. (→ pages 90 and 143.)

A. Maurer and M. Pontil. Empirical Bernstein Bounds and Sample-Variance Penalization. In *Proceedings of the Twenty-Second Conference on Learning Theory*, 2009. (→ page 123.)

V. Mnih, Cs. Szepesvári, and J.-Y. Audibert. Empirical Bernstein stopping. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning*, pages 672–679, 2008. (→ pages 28 and 124.)

R. Munos. Error Bounds for Approximate Policy Iteration. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 560–567, 2003. (→ pages 19, 58, 66, 68, and 70.)

R. Munos. Performance Bounds in L_p-norm for Approximate Value Iteration. *SIAM Journal on Control and Optimization*, 46(2):541–561, 2007. (→ pages 37, 65, 68, and 70.)

R. Munos and Cs. Szepesvári. Finite-Time Bounds for Fitted Value Iteration. *Journal of Machine Learning Research*, 9:815–857, 2008. (→ pages XIV, 8, 19, 37, 58, 59, 68, and 70.)

A. Ng, H. J. Kim, M. Jordan, and S. Sastry. Inverted autonomous helicopter flight via reinforcement learning. In *International Symposium on Experimental Robotics*. MIT Press, 2004. (→ pages VII and 1.)

J. Nunen. A set of successive approximation methods for discounted markovian decision problems. *Zeitschrift für Operations Research*, 20(5):203–208, 1976. (→ page 17.)

O. Pietquin, M. Geist, S. Chandramohan, and H. Frezza-Buet. Sample-Efficient Batch Reinforcement Learning for Dialogue Management Optimization. *ACM Transactions on Speech and Language Processing*, 7(3):7:1–7:21, May 2011. (→ pages VII and 1.)

B. Pires, M. Ghavamzadeh, and Cs. Szepesvári. Cost-sensitive Multiclass Classification Risk Bounds. In *Proceedings of the Thirtieth International Conference on Machine Learning*, 2013. (→ pages 90 and 143.)

D. Precup, R. Sutton, and S. Singh. Eligibility Traces for Off-Policy Policy Evaluation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 759–766, 2000. (→ page 65.)

D. Precup, R. Sutton, and S. Dasgupta. Off-Policy Temporal Difference Learning with Function Approximation. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 417–424, 2001. (→ page 65.)

M. Puterman. *Markov Decision Processes*. Wiley, New York, 1994. (→ page 14.)

M. Puterman and M. Shin. Modified Policy Iteration Algorithms for Discounted Markov Decision Problems. *Management Science*, 24(11), 1978. (→ pages XI, 6, 10, 17, 57, and 142.)

I. Rexakis and M. Lagoudakis. Directed Policy Search Using Relevance Vector Machines. In *IEEE Twenty-Fourth International Conference on Tools with Artificial Intelligence*, pages 25–32, 2012. (→ pages 90 and 144.)

H. Robbins. Some Aspects of the Sequential Design of Experiments. *Bulletin of the American Mathematics Society*, 58:527–535, 1952. (→ pages 25, 104, and 105.)

R. Rubinstein and D. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*. Springer-Verlag, 2004. (→ pages 58, 79, and 81.)

B. Scherrer. Performance Bounds for λ-Policy Iteration and Application to the Game of Tetris. *Journal of Machine Learning Research*, 14:1175–1221, 2013. (→ pages IX, 3, 20, 58, 70, 79, and 80.)

B. Scherrer and B. Lesner. On the Use of Non-Stationary Policies for Stationary Infinite-Horizon Markov Decision Processes. In *NIPS*, pages 1835–1843, 2012. (→ pages 90 and 144.)

B. Scherrer, M. Ghavamzadeh, V. Gabillon, and M. Geist. Approximate Modified Policy Iteration. In *Proceedings of the Twenty Ninth International Conference on Machine Learning*, pages 1207–1214, 2012. (→ page 57.)

B. Scherrer, M. Ghavamzadeh, V. Gabillon, B. Lesner, and M. Geist. Approximate Modified Policy Iteration. *Submitted to Journal of Machine Learning Research*, 2014. (→ page 57.)

P. Schweitzer and A. Seidman. Generalized polynomial approximations in markovian decision processes. *Journal of Mathematical Analysis and Applications*, 110:568–582, 1985. (→ page 34.)

D. I. Simester, P. Sun, and J. Tsitsiklis. Dynamic Catalog Mailing Policies. *Management Science*, 52(5):683–696, May 2006. (→ pages VII and 1.)

R. Sutton. *Temporal credit assignment in reinforcement learning*. PhD thesis, University of Massachusetts Amherst, 1984. (→ pages XI, 6, and 23.)

R. Sutton and A. Barto. *Reinforcement Learning, An introduction*. BradFord Book. The MIT Press, 1998. (→ page 20.)

Cs. Szepesvári. Reinforcement Learning Algorithms for MDPs. In *Wiley Encyclopedia of Operations Research*. Wiley, 2010. (→ pages 19 and 58.)

I. Szita and A. Lőrincz. Learning Tetris Using the Noisy Cross-Entropy Method. *Neural Computation*, 18(12):2936–2941, 2006. (→ pages IX, 4, 79, 80, and 81.)

G. Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219, 1994. (→ pages VII and 1.)

C. Thiéry and B. Scherrer. Building Controllers for Tetris. *International Computer Games Association Journal*, 32:3–11, 2009a. URL `http://hal.inria.fr/inria-00418954`. (→ pages 79 and 81.)

C. Thiéry and B. Scherrer. Improvements on Learning Tetris with Cross Entropy. *International Computer Games Association Journal*, 32, 2009b. URL `http://hal.inria.fr/inria-00418930`. (→ pages IX, 4, 79, 81, 82, and 88.)

C. Thiéry and B. Scherrer. Least-Squares λ-Policy Iteration: Bias-Variance Trade-off in Control Problems. In *Proceedings of the Twenty-Seventh International Conference on Machine Learning*, pages 1071–1078, 2010a. (→ pages 20 and 58.)

C. Thiéry and B. Scherrer. MDPTetris features documentation, 2010b. `http://mdptetris.gforge.inria.fr/doc/feature_functions_8h.html`. (→ pages 81 and 88.)

C. Thiéry and B. Scherrer. Performance Bound for Approximate Optimistic Policy Iteration. Technical report, INRIA, 2010c. (→ page 67.)

J. Tsitsiklis and B. Van Roy. Feature-Based Methods for Large Scale Dynamic Programming. *Machine Learning*, 22:59–94, 1996. (→ pages IX, 3, and 79.)

J. Tsitsiklis and B. Van Roy. An Analysis of Temporal-Difference Learning with Function Approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997. (→ page 71.)

T. Wang, N. Viswanathan, and S. Bubeck. Multiple Identifications in Multi-Armed Bandits. In *Proceedings of the Thirtiethth International Conference on Machine Learning*, volume 28, pages 258–265, 2013. (→ pages 29, 30, 31, 106, 114, and 115.)

Y. Wang, J.-Y. Audibert, and R. Munos. Algorithms for Infinitely Many-Armed Bandits. In *NIPS*, pages 1729–1736, 2008. (→ page 26.)