Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR Sciences et Technologies

Itération sur les Politiques Optimiste et Apprentissage du Jeu de Tetris

THÈSE

présentée et soutenue publiquement le 25 novembre 2010

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1
(spécialité informatique)

par

Christophe Thiéry

Composition du jury

Rapporteurs :	Michèle SEBAG	Directrice de Recherche CNRS, LRI, Université Paris Sud
	Rémi MUNOS	Directeur de Recherche, INRIA Lille Nord Europe
Examinateurs :	Olivier SIGAUD	Professeur, ISIR, UPMC Paris
	Bernard GIRAU	Professeur, Université Henri Poincaré, Nancy 1
	François CHARPILLET	Directeur de Recherche, INRIA Nancy Grand Est
	Bruno SCHERRER	Chargé de Recherche, INRIA Nancy Grand Est

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thloria.

Résumé

Cette thèse s'intéresse aux méthodes d'itération sur les politiques dans l'apprentissage par renforcement à grand espace d'états avec approximation linéaire de la fonction de valeur. Nous proposons d'abord une unification des principaux algorithmes du contrôle optimal stochastique. Nous montrons la convergence de cette version unifiée vers la fonction de valeur optimale dans le cas tabulaire, ainsi qu'une garantie de performances dans le cas où la fonction de valeur est estimée de façon approximative. Nous étendons ensuite l'état de l'art des algorithmes d'approximation linéaire du second ordre en proposant une généralisation de Least-Squares Policy Iteration (LSPI) (Lagoudakis et Parr, 2003). Notre nouvel algorithme, Least-Squares λ Policy Iteration (LSλPI), ajoute à LSPI un concept venant de λ-Policy Iteration (Bertsekas et Ioffe, 1996) : l'évaluation amortie (ou optimiste) de la fonction de valeur, qui permet de réduire la variance de l'estimation afin d'améliorer l'efficacité de l'échantillonnage. LSλPI propose ainsi un compromis biais-variance réglable qui peut permettre d'améliorer l'estimation de la fonction de valeur et la qualité de la politique obtenue.

Dans un second temps, nous nous intéressons en détail au jeu de Tetris, une application sur laquelle se sont penchés plusieurs travaux de la littérature. Tetris est un problème difficile en raison de sa structure et de son grand espace d'états. Nous proposons pour la première fois une revue complète de la littérature qui regroupe des travaux d'apprentissage par renforcement, mais aussi des techniques de type évolutionnaire qui explorent directement l'espace des politiques et des algorithmes réglés à la main. Nous constatons que les approches d'apprentissage par renforcement sont à l'heure actuelle moins performantes sur ce problème que des techniques de recherche directe de la politique telles que la méthode d'entropie croisée (Szita et Lőrincz, 2006). Nous expliquons enfin comment nous avons mis au point un joueur de Tetris qui dépasse les performances des meilleurs algorithmes connus jusqu'ici et avec lequel nous avons remporté l'épreuve de Tetris de la Reinforcement Learning Competition 2008.

Mots-clés: contrôle optimal stochastique, apprentissage par renforcement, programmation dynamique, Processus Décisionnels de Markov, Least-Squares Policy Iteration, λ-Policy Iteration, approximation de la fonction de valeur, compromis biais-variance, fonctions de base, Tetris, méthode d'entropie croisée

Abstract

This thesis studies policy iteration methods with linear approximation of the value function for large state space problems in the reinforcement learning context. We first introduce a unified algorithm that generalizes the main stochastic optimal control methods. We show the convergence of this unified algorithm to the optimal value function in the tabular case, and a performance bound in the approximate case when the value function is estimated. We then extend the literature of second-order linear approximation algorithms by proposing a generalization of Least-Squares Policy Iteration (LSPI) (Lagoudakis et Parr, 2003). Our new algorithm, Least-Squares λ Policy Iteration (LSλPI), adds to LSPI an idea of λ-Policy Iteration (Bertsekas et Ioffe, 1996) : the damped (or optimistic) evaluation of the value function, which allows to reduce the variance of the estimation to improve the sampling efficiency. Thus, LSλPI offers a bias-variance trade-off that may improve the estimation of the value function and the performance of the policy obtained.

In a second part, we study in depth the game of Tetris, a benchmark application that several works from the literature attempt to solve. Tetris is a difficult problem because of its structure and its large state space. We provide the first full review of the literature that includes reinforcement learning works, evolutionary methods that directly explore the policy space and handwritten controllers. We observe that reinforcement learning is less successful on this problem than direct policy search approaches such as the cross-entropy method (Szita et Lőrincz, 2006). We finally show how we built a controller that outperforms the previously known best controllers, and shortly discuss how it allowed us to win the Tetris event of the 2008 Reinforcement Learning Competition.

Keywords: Stochastic optimal control, Reinforcement Learning, Dynamic Programming, Markov Decision Processes, Least-Squares Policy Iteration, λ-Policy Iteration, value function approximation, bias-variance trade-off, feature functions, Tetris, cross-entropy method

Remerciements

Je tiens à remercier toutes les personnes qui ont contribué à la réussite de ce travail. En premier lieu, je remercie lieu Bruno Scherrer pour sa disponibilité quotidienne et son implication. Ses commentaires et ses idées ont été d'une aide précieuse tout au long de cette thèse.

Je rends également hommage à François Charpillet qui m'a accueilli pendant ces trois années dans l'équipe Maia, une équipe dans laquelle j'ai eu la chance de trouver une ambiance de travail sympathique et motivante. Merci à Vincent Chevrier et Christine Bourjot pour leur bonne humeur, ainsi qu'à Alain Dutech, Vincent Thomas, Olivier Buffet et l'ensemble de l'équipe Maia pour leurs nombreux retours toujours riches en enseignements.

Je souhaiterais en outre remercier Michèle Sebag et Rémi Munos qui ont rapporté ce travail, ainsi qu'Olivier Sigaud et Bernard Girau qui ont accepté de faire partie du jury. Je remercie enfin toutes les personnes qui m'ont soutenues au cours de ce travail sur le plan personnel et familial.

Table des matières

Introduction **7**

I Contrôle optimal stochastique **17**

1 Fondements de l'apprentissage par renforcement **19**
1.1 Formalisme des PDM 20
1.1.1 Politique et fonction de valeur 20
1.1.2 Equations de Bellman 20
1.1.3 Fonctions de valeur Q 22
1.2 Algorithmes fondamentaux 24
1.2.1 Value Iteration 24
1.2.2 Policy Iteration 24
1.2.3 Modified Policy Iteration 25
1.2.4 λ-Policy Iteration 26
1.2.5 La notion d'optimisme 28

2 Une vision unifiée **31**
2.1 Unified Policy Iteration 31
2.2 Résultat de convergence 32
2.3 Illustration : Modified λ-Policy Iteration 34

II Le cas approché **41**

3 Apprentissage par renforcement avec approximation **43**
3.1 Borne de performance 43
3.2 Architecture d'approximation linéaire 45
3.3 Approximation linéaire du premier ordre 46
3.3.1 TD(0) 46
3.3.2 TD(λ) 48
3.3.3 Limites des approches du premier ordre 50
3.4 Approximation linéaire du second ordre 50

3.4.1 LSTD et LSTD(λ) . . . 51
3.4.2 LSPE(λ) . . . 52
3.4.3 Approximate λPI . . . 54
3.4.4 LSPI . . . 55

4 LSλPI : Optimisme et compromis biais-variance pour le contrôle optimal 61
4.1 L'algorithme LSλPI . . . 61
4.1.1 Idée générale . . . 62
4.1.2 Méthode de projection du point fixe : LSλTDQ . . . 62
4.1.3 Méthode de minimisation du résidu quadratique : LSλBRQ . . . 65
4.1.4 Least-Squares λ Policy Iteration . . . 67
4.1.5 Cas possible d'une erreur non contrôlée . . . 68
4.2 Expériences . . . 70

III Etude de cas : le jeu de Tetris 77

5 Etat de l'art des travaux sur Tetris 79
5.1 Le problème de Tetris . . . 79
5.2 Principales approches . . . 80
5.2.1 Approches par apprentissage par renforcement . . . 82
5.2.2 Approches d'optimisation générale . . . 84
5.2.3 Contrôleurs réglés manuellement . . . 84
5.3 Difficulté de comparer les joueurs artificiels . . . 85
5.3.1 Spécification du jeu . . . 85
5.3.2 Grande variance des scores à Tetris . . . 85
5.3.3 Subtilités d'implémentation . . . 86
5.4 La méthode d'entropie croisée . . . 87

6 Nouveaux résultats sur Tetris 93
6.1 Contrôle optimal exact . . . 93
6.2 Approximation linéaire : LSλPI . . . 93
6.3 Méthode d'entropie croisée . . . 95
6.4 Vers un contrôleur performant . . . 98
6.5 Reinforcement Learning Competition 2008 . . . 100

Conclusion générale 103

Annexes 109

A Preuve de la borne de performance 111

Bibliographie **115**

Table des figures

1 Représentation schématique du problème de l'apprentissage supervisé. 11
2 Représentation schématique du problème de l'apprentissage non supervisé. 12
3 Représentation schématique du problème de l'apprentissage par renforcement. 13

1.1 L'interface agent-environnement selon Sutton et Barto (1998). 19
1.2 Vue intuitive de la notion d'optimisme dans la partition de l'espace des fonctions de valeur selon leur politique gloutonne. 28

2.1 Généralisation des algorithmes classiques par MλPI. 35
2.2 L'environnemnt utilisé pour les expériences de la figure 2.3. 36
2.3 Vitesse de convergence de MλPI en fonction de λ et m. 37
2.4 Vitesse de convergence de MλPI pour un ensemble plus fin de valeurs de λ et m. 38

3.1 Représentation schématique de la projection de $T_\pi \widehat{V}$ sur l'espace d'approximation. 47
3.2 Représentation schématique de la projection de $T_\lambda \widehat{V}$ sur l'espace d'approximation. 49
3.3 Illustration du compromis biais-variance de λPI dans la partition des politiques gloutonnes. 55
3.4 Représentation schématique des deux méthodes LSTDQ et LSBRQ. 56
3.5 Vue d'ensemble des algorithmes d'approximation linéaire mentionnés. 58

4.1 Représentation schématique des deux méthodes LSλTDQ et LSλBRQ. 64
4.2 Convergence de LSλTDQ en fonction de λ et γ. 69
4.3 Convergence de LSλBRQ en fonction de λ et γ. 69
4.4 Ratio des vitesses de convergence de LSλTDQ et LSλBRQ. 70
4.5 Représentation du problème de la chaîne d'états étudié. 71
4.6 Chaîne d'états : distance de la fonction de valeur courante par rapport à la valeur optimale pour plusieurs valeurs de λ. 72
4.7 Chaîne d'états : distance de la politique courante par rapport à la politique optimale pour plusieurs valeurs de λ. 73
4.8 Chaîne d'états : observation d'une propriété de convergence spécifique à LSλTDQ. 74

5.1 Illustration du problème de Tetris. 80
5.2 Principe d'un contrôleur à une pièce. 81
5.3 Principe d'un contrôleur à deux pièces. 81
5.4 Représentation graphique de la méthode d'entropie croisée bruitée. 89

6.1 Score moyen de 100 parties de Tetris pour différentes valeurs de λ à chaque itération de LSλPI. 94
6.2 Notre implémentation de l'expérience de Szita et Lőrincz (2006). 96
6.3 Détail des 10 exécutions de notre implémentation de chaque expérience de Szita et Lőrincz (2006). 97
6.4 Evolution du score moyen de 30 parties avec la méthode d'entropie croisée bruitée pour différents ensembles de fonctions de base. 99

Introduction

Les êtres vivants disposent naturellement de capacités remarquables. Ils sont capables de percevoir leur environnement, de reconnaître et de classer des formes ou des signaux de toutes sortes, de communiquer, d'interagir avec leur entourage, de se mouvoir ou encore de raisonner. Au fur et à mesure de l'expérience qu'ils acquièrent, ils savent également améliorer leur comportement et leurs réactions. Face à une situation donnée, ils sont ainsi capables d'exploiter les connaissances qu'ils ont obtenues lors de situations similaires vécues dans le passé. Ces aptitudes intellectuelles propres aux êtres vivants sont autant de défis pour l'intelligence artificielle, l'un des domaines de la recherche en informatique.

Cette thèse s'intéresse à la manière de doter les machines de telles capacités. En particulier, nous nous appuyons sur l'apprentissage par renforcement, une composante de l'intelligence artificielle dans laquelle un système cherche à apprendre à choisir les bonnes décisions. On souhaite qu'une décision entraînant une sensation agréable à plus ou moins long terme soit davantage choisie par la suite dans des circonstances analogues qu'une action suivie d'une réponse moins favorable. Pour cela, l'apprentissage par renforcement propose un formalisme général qui permet de modéliser de nombreux types de problèmes, ainsi que des outils algorithmiques qui visent à les résoudre. L'apprentissage par renforcement possède en outre d'autres caractéristiques intéressantes. Le système apprenant est autonome, c'est-à-dire qu'il est capable d'apprendre seul un comportement face à une situation inconnue. De plus, il évolue dans un environnement incertain : les résultats de ses actions, sur lesquels il s'appuie pour construire son expérience, sont probabilistes. Enfin, l'apprentissage par renforcement peut être adaptatif : si les règles du problème changent à un moment donné, le système est capable de réapprendre et de mettre à jour sa stratégie au fur et à mesure qu'il continue à acquérir de l'expérience.

Contexte scientifique

Avant de détailler nos travaux sur l'apprentissage par renforcement, nous proposons un bref panorama de l'intelligence artificielle.

L'intelligence artificielle

La question de créer ce que l'on pourrait appeler une « machine intelligente » peut se décliner en quelques grandes thématiques que nous mentionnons ci-dessous. Une introduction générale à l'intelligence articielle est proposée dans l'ouvrage de Russell *et al.* (1996).

- **Perception et reconnaissance :** il s'agit de la capacité de la machine à analyser les données provenant d'un ensemble de capteurs (visuels, sonores ou autres) dans le but de construire une représentation de son environnement. La reconnaissance de la parole, la reconnaissance de formes et la stéréovision sont des problèmes qui font appel à cette capacité de perception. Dans le domaine de la vision, la reconnaissance de formes permet par exemple d'identifier des visages sur des photographies. La reconnaissance de la parole est également un sujet très actif de la recherche en intelligence artificielle et trouve de nombreuses applications, notamment dans les interfaces vocales, la transcription automatique de la parole ou encore la téléphonie mobile.
- **Représentation des connaissances :** il s'agit ici de doter la machine d'une représentation symbolique d'un ensemble d'informations et de connaissances sur le monde. On souhaite donner à la machine la faculté de représenter des concepts, les propriétés de ces concepts et les relations entre eux. On cherche aussi à exprimer des situations et des événements dans le temps, avec leurs causes et leurs conséquences. La notion d'ontologie permet ainsi de modéliser un domaine et de raisonner dessus en représentant formellement les concepts et les relations de ce domaine.

- **Traitement automatique des langues :** le traitement automatique des langues cherche à donner à la machine la capacité de lire et de comprendre les langues parlées par l'homme. Contrairement aux langages informatiques qui sont conçus de manière très cadrée et de façon à être interprétés par la machine à partir de règles simples et précises, les langues naturelles sont intrinsèquement riches, complexes et pourvues d'ambiguïtés. À court terme, les applications immédiates de la recherche en traitement automatique des langues sont la synthèse d'informations et la traduction automatique de textes. À plus long terme, l'objectif serait de rendre un système capable d'acquérir de la connaissance de façon autonome en lisant simplement des textes existants.
- **Planification :** la planification consiste pour un agent intelligent à déterminer la manière d'atteindre un but, comme par exemple calculer son chemin dans un environnement. Il est nécessaire pour l'agent de connaître une représentation du monde et de pouvoir prédire la manière dont ses actions vont l'influencer, afin de faire les choix qui permettent de le mener au but. Dans la planification classique, les actions de l'agent influent de façon déterministe sur l'environnement. D'autres types de planification prennent en compte une incertitude dans les effets des actions et imposent à l'agent de recalculer un plan si ses observations ne correspondent pas à ses prédictions. La planification multi-agents fait quant à elle appel à la coordination ou à la compétition de plusieurs agents pour atteindre un but qui peut être collectif.
- **Apprentissage automatique :** l'apprentissage automatique (Mitchell, 1997) désigne le développement, l'analyse et l'implémentation de systèmes informatiques capables de s'améliorer à partir d'un ensemble d'exemples ou à partir de leur propre expérience. L'objectif peut être d'apprendre à catégoriser des données (apprentissage supervisé ou non supervisé) ou à prendre les meilleures décisions possibles en se basant sur l'expérience pour résoudre un problème (apprentissage par renforcement).

Nous nous intéressons dans cette thèse à permettre à un système qui prend des décisions séquentielles d'apprendre à effectuer les meilleurs choix à l'aide de son expérience. Ce travail s'inscrit donc essentiellement dans la thématique de l'apprentissage automatique, même si en raison du caractère séquentiel des prises de décision, notre propos abordera également quelques notions de planification. Nous présentons maintenant un aperçu de l'apprentissage automatique.

L'apprentissage automatique

Comment peut-on permettre à un ordinateur d'apprendre de façon autonome ? L'idée naturelle est de s'inspirer du processus d'apprentissage des êtres vivants. On peut considérer que, chez un être humain, l'apprentissage se décline de trois manières différentes que nous détaillons ci-dessous en nous appuyant sur deux exemples de processus d'apprentissage : un enfant et un joueur d'échecs.

L'apprentissage supervisé

D'abord, un être vivant apprend grâce aux connaissances et aux compétences qui lui sont explicitement transmises. Ce mode d'apprentissage est appelé **apprentissage supervisé**. Dans le cas d'un enfant, sa famille, son entourage et ses enseignants lui expliquent directement un grand nombre de choses comme les nombres, la définition des mots ou encore la politesse. Dans un jeu comme les échecs, l'apprentissage supervisé consiste à expliquer au joueur novice le coup qu'il doit jouer dans telle ou telle situation, ce qui constitue une connaissance déterminante, surtout au début et à la fin des parties.

En intelligence artificielle, le problème est modélisé (voir figure 1) comme l'apprentissage d'une base d'exemples, où chaque exemple associe à une situation donnée une étiquette (ou catégorie). Une fois cette base d'exemples acquise, l'objectif du sytème est de savoir attribuer une bonne étiquette à une nouvelle situation, inconnue jusqu'ici de la base d'apprentissage, mais éventuellement proche de certaines situations déjà connues. Dans le cas de l'enfant, les exemples peuvent être des mots étiquetés par leur définition et, dans le cas du joueur d'échecs, les exemples sont des situations précises du jeu étiquetées par les coups à jouer dans ces situations. Si l'apprentissage est performant, le système saura quelle étiquette choisir lorsqu'il sera confronté à une situation inédite.

Le système doit donc être capable de généraliser ce qu'il apprend sur la base d'exemples. Cette capacité de généralisation est le cœur du problème de l'apprentissage supervisé : si le sytème apprend parfaitement

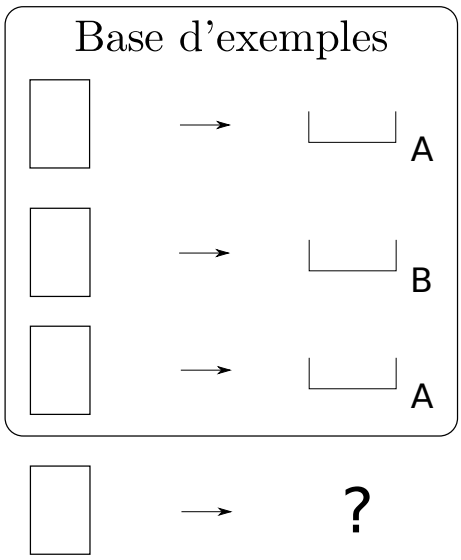


FIGURE 1 – Représentation schématique du problème de l'apprentissage supervisé. On dispose d'une base d'exemples étiquetés, c'est-à-dire que chaque exemple attribue une étiquette à une situation. Le système cherche à apprendre cette base d'exemples afin de savoir étiqueter de futures situations.

la base d'apprentissage sans savoir exploiter ces connaissances, il ne saura pas étiqueter correctement une nouvelle situation. On parle alors d'apprentissage par cœur ou de sur-apprentissage. Apprendre par cœur des coups d'échecs sans comprendre leur motivation ne permettra pas au joueur de s'en sortir dans des situations similaires mais légèrement différentes. De plus, les exemples de la base d'apprentissage peuvent être bruités, voire comporter des éléments incohérents. Il est donc important de ne pas accorder une confiance absolue à la base d'apprentissage.

Les techniques de classification supervisée et de régression permettent d'effectuer de l'apprentissage supervisé. Les étiquettes sont des valeurs discrètes dans le cas de la classification (on cherche alors à construire un classifieur) et continues dans le cas de la régression (on cherche alors à construire un approximateur de fonctions). Parmi ces approches, on peut citer les réseaux de neurones de type perceptrons multi-couches (Bishop, 1996), les arbres de décision (Breiman *et al.*, 1984) ou encore les machines à vecteurs supports (SVM) (Andrew, 2000).

L'apprentissage non supervisé

Une part de l'apprentissage est également effectuée de façon totalement autonome : il s'agit de **l'apprentissage non supervisé**. L'enfant est capable de découvrir le monde en partie par lui-même. Par exemple, avant même d'apprendre à parler, un bébé apprend de façon autonome à faire des mouvements et à relier ces mouvements à ses perceptions (il s'agit du développement dit sensori-moteur). Dans le cas du jeu d'échecs, l'apprentissage non supervisé consiste pour le joueur à découvrir par lui-même des types de situations de jeu, et par la suite à savoir les identifier au fur et à mesure de son expérience : une situation de fin de partie, l'occupation géographique d'une certaine zone du plateau, une structure particulière des pions, etc. Il ne s'agit pas ici de choisir une bonne action à jouer, ni même d'évaluer si la situation est bonne ou mauvaise, mais de découvrir des caractéristiques qui vont permettre de classer les situations de façon pertinente.

Du point de vue de l'intelligence artificielle, l'apprentissage non supervisé (voir figure 2) consiste à apprendre une base d'exemples, où chaque exemple est cette fois une situation non étiquetée. C'est au système de découvrir des étiquettes appropriées et de définir des critères pour les attribuer. Comme dans le cas de l'apprentissage supervisé, il convient de généraliser de façon pertinente ce qui est observé dans la base d'apprentissage et de ne pas faire de sur-apprentissage. Le joueur d'échecs doit pouvoir reconnaître une situation de jeu inédite grâce aux informations qu'il possède sur des situations similaires

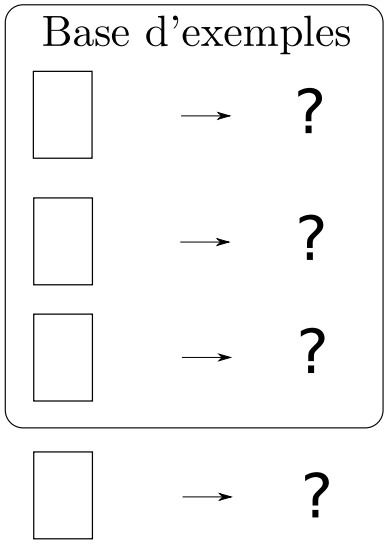


FIGURE 2 – Représentation schématique du problème de l'apprentissage non supervisé. On dispose d'une base d'exemples, où chaque exemple est une situation seule (non étiquetée). Le système cherche à classifier ces exemples dans des catégories qu'il découvre lui-même.

déjà rencontrées.

Les techniques d'apprentissage non supervisé incluent les algorithmes de *clustering* tels que K-means (Kanungo *et al.*, 2002), l'analyse en composantes indépendantes (Hyvärinen, 2001) et les cartes auto-organisatrices (Kohonen, 1989; Ritter *et al.*, 1992) qui sont une forme de réseaux de neurones.

L'apprentissage par renforcement

Enfin, un troisième mode d'apprentissage, nommé **apprentissage par renforcement** (Sutton et Barto, 1998), consiste à apprendre de façon autonome non pas à classer des situations, mais à effectuer des actions dans des situations (voir figure 3). Un être vivant effectue par lui-même des actions qui sont suivies d'un effet positif ou négatif appelé récompense. Grâce à cette récompense, il acquiert alors de l'expérience et finit par connaître les choix qu'il est souhaitable de faire pour obtenir le maximum d'effets positifs. Par exemple, un enfant apprend ce qu'il a le droit de faire ou non en essayant spontanément ce dont il a envie : s'amuser dans le calme, ou en faisant du bruit ou des bêtises. L'effet positif ou négatif est une réaction de la part de ses parents. Il semble raisonnable de penser qu'une grande part de l'éducation provienne de ce type d'apprentissage. Aux échecs, dans une situation d'incertitude, le joueur peut essayer un coup et se rendre compte plus tard dans la partie que c'était un bon ou un mauvais choix. Dans les deux cas, il acquiert de l'expérience en faisant un essai et en obtenant une récompense positive ou négative, ce qui lui permettra d'améliorer ses choix futurs dans des situations similaires.

L'apprentissage par renforcement peut être adaptatif. Aux échecs, si l'adversaire se met à changer de stratégie, le joueur va pouvoir s'en rendre compte en réalisant que des actions qu'il croyait bonnes sont devenues moins bonnes. À l'inverse, de mauvaises actions peuvent devenir bonnes si la conséquence des actions a changé. Pour obtenir les meilleures récompenses possibles tout en apprenant les conséquences des actions, le système va devoir trouver de bons compromis.

- Est-il important d'obtenir immédiatement une récompense élevée, au risque d'avoir de nombreuses récompenses négatives plus tard ? Autrement dit, préfère-t-on agir sur le court terme ou sur le long terme ?
- Faut-il explorer l'environnement au risque d'effectuer de mauvaises actions, ou faut-il exploiter en priorité les actions connues comme bonnes, au risque de ne jamais découvrir les meilleures ? Ce dilemme exploration-exploitation est l'un des problèmes auquel un système qui apprend par renforcement se confronte.

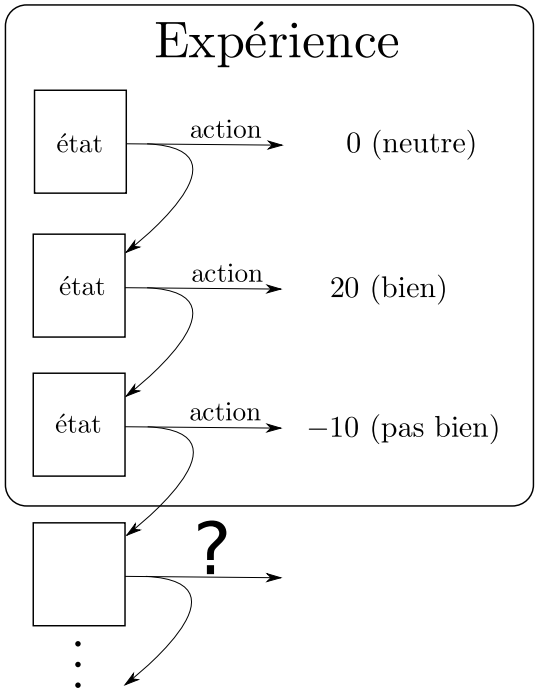


FIGURE 3 – Représentation schématique du problème de l'apprentissage par renforcement. Un agent évolue dans un environnement. Lorsqu'il effectue une action depuis un certain état, il obtient une certaine récompense et arrive dans un nouvel état. À partir de son expérience constituée par une ou plusieurs séquences d'états visités, d'actions effectuées dans ces états et de récompenses obtenues, l'objectif est d'apprendre à choisir dans n'importe quel état les actions qui vont maximiser le cumul des récompenses sur le long terme.

– Lorsque le nombre de situations possibles est très élevé, comment répercuter l'apprentissage de l'effet d'une action non pas uniquement sur la situation qui a été rencontrée, mais sur l'ensemble des situations similaires ? Cette question, liée à la taille du problème à résoudre, fait appel à des techniques d'estimation et d'approximation de fonctions et rejoint la problématique de la généralisation mentionnée dans les cas des apprentissages supervisé et non supervisé.

Problématique et contributions

Même s'il est probable que ces trois types d'apprentissage interviennent simultanément dans un système apprenant idéal ou chez les êtres vivants, nous nous focalisons dans cette thèse sur l'apprentissage par renforcement, et plus particulièrement sur le traitement des problèmes à grand espace d'états. Comment un système peut-il améliorer ses actions lorsque les récompenses qu'il obtient sont probabilistes ? Comment peut-il apprendre les conséquences d'actions dont l'effet ne sera observé que bien plus tard ? Comment peut-il maximiser ses récompenses au fil de l'expérience accumulée, alors que le nombre de situations possibles est tel qu'il ne retrouvera que rarement voire jamais deux fois la même situation ?

Nous étudions des techniques d'approximation de fonctions qui peuvent être utilisées pour apprendre efficacement à partir d'expériences imprécises, incomplètes et bruitées. Nous nous intéressons plus spécifiquement dans cette thèse à deux algorithmes de la littérature. Le premier d'entre eux est λ-Policy Iteration (λPI) (Bertsekas et Ioffe, 1996), une technique qui généralise les deux algorithmes classiques du contrôle optimal stochastique que sont Value Iteration et Policy Iteration (Puterman, 1994) en introduisant une notion d'optimisme réglable. L'optimisme consiste à évaluer la politique courante de façon amortie (incomplète) et de passer immédiatement à la politique suivante. Nous proposons une généralisation de ces algorithmes qui permet d'exprimer la notion d'optimisme de manière unifiée. Dans le cas exact, nous montrons la convergence de cet algorithme unifié et nous étudions expérimentalement une application sur un problème de navigation discrète (Thiery et Scherrer, 2009c). Dans le cas approché, nous montrons une garantie de performance théorique.

Le second algorithme sur laquelle nous nous penchons est Least-Squares Policy Iteration (LSPI) (Lagoudakis et Parr, 2003), une approche d'itération sur les politiques avec approximation linéaire du second ordre, c'est-à-dire qui exploite de façon efficace la connaissance constituée par les échantillons qui sont à sa disposition. La contribution essentielle de cette thèse est de proposer un nouvel algorithme intitulé LSλPI (Thiery et Scherrer, 2010) : il s'agit d'une généralisation de LSPI qui approxime λPI. LSλPI regroupe les avantages de ces deux approches. Notre proposition permet d'ajouter la notion d'optimisme à LSPI et de faire un compromis entre la variance de l'estimation de la fonction de valeur et le biais lié à l'optimisme. Si l'optimisme permet de réduire la variance en introduisant une certaine prudence dans l'estimation de la fonction de valeur, il constitue également un biais dans la mesure où l'on change de politique avant d'avoir entièrement évalué la politique précédente. Nous montrons expérimentalement que ce compromis biais-variance peut permettre d'améliorer la qualité de l'estimation et les performances de la politique obtenue. En résumé, il s'agit à notre connaissance du premier algorithme qui cumule les caractéristiques suivantes :

– l'échantillonnage efficace : on réalise une approximation dite du second ordre,
– l'optimisme dans l'évaluation : on n'attend pas que la politique soit entièrement évaluée avant de la changer,
– la présence d'un paramètre λ qui réalise un compromis biais-variance lors de l'estimation de la fonction de valeur,
– l'évaluation off-policy : on peut évaluer une politique autre que celle utilisée pour générer les échantillons.

Dans un second temps, nous étudions de façon détaillée une application : le jeu de Tetris, qui est un problème difficile à résoudre en raison de sa structure et de son très grand nombre d'états. Nous explorons l'état de l'art des différentes approches permettant de traiter le problème : l'apprentissage par renforcement, les techniques d'optimisation directe de la politique et les algorithmes réglés de façon manuelle. Nous proposons la première revue qui regroupe ces trois types d'approches (Thiery et Scherrer, 2009a), en dressant notamment la liste des fonctions de base utilisées par les travaux dont nous avons connaissance. Nous mettons en évidence le fait que la comparaison de différents travaux doit être faite avec soin en raison de la grande variance des scores à Tetris et de certaines subtilités d'implémentation qui

peuvent avoir un impact considérable sur les résultats. En nous appuyant sur cette étude bibliographique, nous proposons ensuite différents résultats expérimentaux. Nous montrons que, sur une taille réduite du problème, l'apprentissage par renforcement permet d'obtenir la fonction de valeur optimale. Sur la taille normale du jeu, nous utilisons LSλPI et constatons que ce dernier a besoin de beaucoup moins d'échantillons que LSPI pour obtenir des performances similaires. Par ailleurs, en associant les points forts de plusieurs techniques diverses (la méthode d'entropie croisée (Szita et Lőrincz, 2006) et les bonnes connaissances expertes de Dellacherie (Fahey, 2003)), et en améliorant ces points forts, nous avons mis au point un joueur qui réalise à notre connaissance les meilleures performances à ce jour (Thiery et Scherrer, 2009b). Notre joueur a ainsi remporté l'épreuve de Tetris dans la Reinforcement Learning Competition (compétition d'apprentissage par renforcement) en 2008.

Plan de la suite du mémoire

Notre démarche est structurée de la façon suivante.

- Dans le premier chapitre, nous introduisons dans le cas exact le cadre des Processus Décisionnels de Markov, qui permet de formaliser le problème de l'apprentissage par renforcement, et nous présentons les principaux algorithmes du contrôle optimal stochastique ainsi que λPI (Bertsekas et Ioffe, 1996).
- Dans le chapitre 2, nous unifions ces algorithmes en introduisant une méthode générale, Unified Policy Iteration, qui permet d'exprimer la notion d'optimisme. Nous proposons une application sur un problème de type navigation discrète.
- Le chapitre 3 introduit le cas approximatif. Nous fournissons d'abord une borne de performance sur les versions approximatives de Unified Policy Iteration. Puis nous dressons un état de l'art des principales approches d'apprentissage par renforcement avec approximation linéaire du second ordre, lesquelles exploitent de façon efficace l'expérience accumulée, notamment LSPI (Lagoudakis et Parr, 2003).
- Dans le chapitre 4, nous proposons un nouvel algorithme du second ordre, LSλPI, qui regroupe plusieurs caractéristiques intéressantes des approches de la littérature. Ces caractéristiques incluent l'apprentissage d'une bonne politique à partir de l'expérience issue d'autres politiques, et un compromis réglable entre la variance des estimations et le biais causé par l'optimisme.
- Le chapitre 5 étudie en détail le problème de Tetris, une application de référence dans la communauté de l'apprentissage par renforcement mais également traitée par des algorithmes de type évolutionnaires.
- Enfin, le chapitre 6 décrit les résultats que nous avons obtenus sur Tetris avec différentes approches.

Première partie

Contrôle optimal stochastique

Chapitre 1

Fondements de l'apprentissage par renforcement

L'apprentissage par renforcement (Sutton et Barto, 1998) considère un agent informatique devant prendre des décisions en interagissant avec son environnement de manière à maximiser un signal de récompense sur le long terme. A chaque instant t, l'agent est dans un état s_t et effectue une action a_t. Il obtient ensuite une récompense $r_{t+1} \in \mathbb{R}$ et arrive dans l'état s_{t+1}. L'objectif de l'agent est de déterminer dans chaque état l'action qui permet d'obtenir le meilleur cumul de récompenses à venir. La figure 1.1 schématise l'interaction entre l'agent et son environnement.

Le problème de l'apprentissage par renforcement consiste à exploiter l'expérience accumulée jusqu'à présent pour rechercher les actions qui permettent de maximiser les récompenses futures. Cependant, nous allons dans un premier temps supposer que le modèle des interactions est connu, c'est-à-dire que pour un problème donné, l'agent connaît à l'avance, en fonction des actions possibles, les probabilités qui régissent les transitions entre états et les récompenses obtenues. Autrement dit, l'agent possède un modèle qui lui fournit une connaissance complète de la dynamique des transitions et des récompenses et, au lieu de se baser sur l'expérience passée, il effectue des calculs à partir de ce modèle afin de déterminer les actions à effectuer selon les états. Dans le réel problème de l'apprentissage par renforcement, que nous traiterons à partir du chapitre 3, l'agent ne connaît pas le modèle et doit se contenter d'exploiter l'expérience qu'il a acquise. Lorsque le modèle est connu, on parle de **contrôle optimal stochastique** plutôt que d'apprentissage par renforcement. Dans ce chapitre, nous introduisons le formalisme du contrôle optimal stochastique et les principales notations qui seront utilisées dans ce mémoire. Nous présentons également quelques algorithmes de contrôle optimal stochastique qui sont le fondement de l'apprentissage par renforcement.

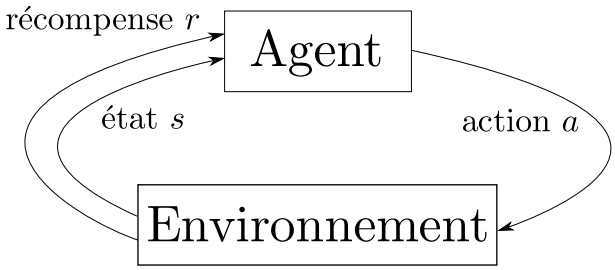


FIGURE 1.1 – L'interface agent-environnement selon Sutton et Barto (1998). A chaque pas de temps, l'agent effectue une action a sur l'environnement, reçoit en retour une récompense r et arrive dans un nouvel état s.

1.1 Formalisme des PDM

Le cadre des Processus Décisionnels de Markov (PDM) permet de formaliser l'apprentissage par renforcement en définissant la manière dont l'agent interagit avec son environnement. Un PDM est la modélisation d'un problème donné dans le contexte du contrôle optimal stochastique. Un PDM est défini comme un tuple $< \mathcal{S}, \mathcal{A}, P, R, \gamma >$ où :

- $\mathcal{S}$ est l'espace des états ;
- $\mathcal{A}$ est l'espace des actions ;
- P est la fonction de transition : $P(s, a, s')$ est la probabilité d'arriver dans l'état s' sachant que l'on est dans l'état s et que l'on effectue l'action a ;
- R est la fonction de récompense : $R(s, a, s') \in \mathbb{R}$ est la récompense reçue en effectuant l'action $a \in \mathcal{A}$ depuis l'état $s \in \mathcal{S}$ et en arrivant dans l'état s' ; on utilisera la notation simplifiée $R(s, a)$ pour désigner la récompense moyenne d'un couple état-action : $R(s, a) = \sum_{s' \in \mathcal{S}} P(s, a, s') R(s, a, s')$;
- $\gamma \in [0, 1]$ est un facteur d'actualisation qui détermine l'influence des récompenses futures.

La fonction de transition P et la fonction de récompense R constituent le modèle du PDM. Comme expliqué précédemment, nous supposons dans ce chapitre et dans le chapitre 2 que la fonction de transition P et la fonction de récompense R sont connues. Pour un PDM donné, l'objectif est de déterminer pour chaque état la meilleure action possible. Pour cela, le contrôle optimal stochastique fait appel à la notion de politique.

1.1.1 Politique et fonction de valeur

Une **politique** représente le choix d'une action à effectuer dans un état donné. Elle est définie de manière générale comme une fonction $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, où $\pi(s, a)$ désigne la probabilité d'effectuer l'action a depuis l'état s. Cependant, on s'intéressera uniquement à des politiques déterministes, c'est-à-dire des politiques avec lesquelles une seule action peut être choisie dans chaque état. On considèrera la notation $\pi : \mathcal{S} \to \mathcal{A}$, où $\pi(s)$ est l'action à effectuer dans l'état s.

On cherche ainsi à déterminer la meilleure politique possible. Pour définir ce qui est une bonne politique, nous avons besoin d'un critère qui quantifie la qualité d'une politique. Bien que différents critères existent, le critère le plus courant, et que nous allons utiliser tout au long de ce mémoire, est la fonction de valeur. La **valeur** d'une politique π est la fonction $V^\pi : \mathcal{S} \to \mathbb{R}$ qui associe à chaque état l'espérance du cumul des récompenses que la politique π obtient à partir de cet état :

$$V^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \middle| s_0 = s, a_t = \pi(s_t) \right] \tag{1.1}$$

Le facteur d'actualisation $\gamma \in [0, 1]$ permet de diminuer l'importance des récompenses lointaines. Si la dynamique du système est telle que la probabilité d'atteindre un état terminal [1] en un temps fini est égale à 1, alors γ peut être égal à 1. Sinon, γ doit être inférieur à 1 pour éviter que la somme diverge vers l'infini.

1.1.2 Equations de Bellman

Une caractéristique primordiale des PDM est que la dynamique des états vérifie la **propriété de Markov**, c'est-à-dire que, dans un état s, les probabilités de transition vers l'état suivant s' dépendent uniquement de s et pas de l'historique des états visités plus tôt ni des actions effectuées. La propriété de Markov est essentielle dans l'apprentissage par renforcement car elle permet d'écrire l'équation de Bellman que nous présentons maintenant et qui est le fondement des algorithmes liés aux PDM.

L'équation de Bellman (Bellman, 1957) est une équation récursive qui établit une propriété fondamentale de la fonction de valeur. Elle s'obtient en développant la définition de la fonction de valeur

1. Un état s est dit terminal si une fois cet état atteint, l'agent y reste indéfiniment et n'obtient plus aucune récompense quelle que soit sa politique. s est terminal si pour toute action a, $P(s, a, s) = 1$ et $R(s, a, s) = 0$.

(équation (1.1)) d'une politique π pour tout état s :

$$\begin{aligned}
V^\pi(s) &= E\left[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t,s_{t+1})\middle| s_0=s, a_t=\pi(s_t)\right] \\
&= E\left[R(s_0,a_0,s_1)+\gamma\sum_{t=0}^{\infty}\gamma^t R(s_{t+1},a_{t+1},s_{t+2})\middle| s_0=s, a_t=\pi(s_t)\right] \\
&= \sum_{s'\in\mathcal{S}} P(s,\pi(s),s')\left(R(s,\pi(s),s')+\gamma E\left[\sum_{t=0}^{\infty}\gamma^t R(s_{t+1},a_{t+1},s_{t+2})\middle| s_0=s, s_1=s', a_t=\pi(s_t)\right]\right) \\
&\quad \text{(d'après la loi de Bayes)} \\
&= \sum_{s'\in\mathcal{S}} P(s,\pi(s),s')\left(R(s,\pi(s),s')+\gamma E\left[\sum_{t=0}^{\infty}\gamma^t R(s_{t+1},a_{t+1},s_{t+2})\middle| s_1=s', a_{t+1}=\pi(s_{t+1})\right]\right) \\
&\quad \text{(car d'après la propriété de Markov, les probabilités de transition à partir de l'état } s_1 \\
&\quad \text{ne dépendent pas de } s_0) \\
&= \sum_{s'\in\mathcal{S}} P(s,\pi(s),s')\left(R(s,\pi(s),s')+\gamma E\left[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t,s_{t+1})\middle| s_0=s', a_t=\pi(s_t)\right]\right) \\
&= R(s,\pi(s))+\gamma\sum_{s'\in\mathcal{S}} P(s,\pi(s),s')V^\pi(s').
\end{aligned}$$

Cette équation récursive énonce que la valeur d'un état dépend de la récompense immédiate et de la valeur des états suivants. Elle est le fondement de nombreux algorithmes permettant de calculer une fonction de valeur ou une politique. On peut la réécrire de manière vectorielle, en considérant V^π comme un vecteur de taille $|\mathcal{S}|$:

$$V^\pi = R_\pi + \gamma P_\pi V^\pi \tag{1.2}$$

où R_π est le vecteur des récompenses moyennes de chaque état en suivant la politique π :

$$R_\pi = \begin{pmatrix} R(s_1,\pi(s_1)) \\ \vdots \\ R(s_{|\mathcal{S}|},\pi(s_{|\mathcal{S}|})) \end{pmatrix}$$

et P_π est la matrice de transition de la chaîne de Markov induite par la politique π :

$$P_\pi = \begin{pmatrix} P(s_1,\pi(s_1),s_1) & \dots & P(s_1,\pi(s_1),s_{|\mathcal{S}|}) \\ \vdots & & \vdots \\ P(s_{|\mathcal{S}|},\pi(s_{|\mathcal{S}|}),s_1) & \dots & P(s_{|\mathcal{S}|},\pi(s_{|\mathcal{S}|}),s_{|\mathcal{S}|}) \end{pmatrix}.$$

On introduit également l'opérateur de Bellman T_π, défini pour tout vecteur V par

$$T_\pi V = R_\pi + \gamma P_\pi V.$$

L'équation de Bellman (1.2) s'écrit donc de manière condensée $V^\pi = T_\pi V^\pi$. Cet opérateur T_π est contractant de facteur γ pour la norme infinie [2] (Puterman, 1994), c'est-à-dire que pour tous vecteurs V et V', on a $\|TV - TV'\|_\infty \leq \gamma\|V - V'\|_\infty$. L'opérateur T_π admet comme unique point fixe la fonction de valeur V^π. Ainsi, V^π est la seule fonction de valeur qui vérifie l'équation de Bellman. On peut aussi voir que T_π est un opérateur monotone, c'est-à-dire que $V \leq V' \Rightarrow T_\pi V \leq T_\pi V'$. En effet,

$$\begin{aligned}
V \leq V' &\Rightarrow \gamma P_\pi V \leq \gamma P_\pi V' \quad \text{car } P_\pi \text{ ne possède que des éléments positifs ou nuls} \\
&\Rightarrow R_\pi + \gamma P_\pi V \leq R_\pi + \gamma P_\pi V' \\
&\Rightarrow T_\pi V \leq T_\pi V'
\end{aligned}$$

2. La norme infinie, notée $\|\cdot\|_\infty$, est l'élément maximal d'un vecteur en valeur absolue, autrement dit : $\|V\|_\infty = \max|V| = \max_s |V(s)|$.

Politique et fonction de valeur optimale

On note V^* la fonction de valeur optimale, qui associe à chaque état la meilleure espérance possible des récompenses :

$$\forall s \in \mathcal{S} \qquad V^*(s) = \max_{\pi} V^{\pi}(s).$$

Il peut exister plusieurs politiques optimales, qui partagent alors cette fonction de valeur. La fonction de valeur optimale vérifie elle aussi une équation récursive, **l'équation d'optimalité de Bellman** (Bellman, 1957) :

$$\forall s \in \mathcal{S} \qquad V^*(s) = \max_{a} \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s,a,s') V^*(s') \right). \tag{1.3}$$

Là aussi, on peut introduire un opérateur, noté T et défini pour tout vecteur V par

$$\forall s \in \mathcal{S} \qquad [TV](s) = \max_{a} \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s,a,s') V(s') \right).$$

L'équation (1.3) peut se réécrire de manière condensée : $V^* = TV^*$. L'opérateur T est contractant de facteur γ pour la norme infinie (Puterman, 1994) et son unique point fixe est la fonction de valeur optimale V^*. Comme T_π, T est un opérateur monotone : si $V \leq V'$, alors $TV \leq TV'$.

Les opérateurs T^π et T permettent notamment d'exprimer le fait qu'une politique soit gloutonne par rapport à une fonction de valeur. Pour toute fonction de valeur V, on appelle **politique gloutonne** par rapport à V une politique π définie par

$$\forall s \in \mathcal{S} \qquad \pi(s) \in \arg\max_{a \in \mathcal{A}} \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s,a,s') V(s') \right). \tag{1.4}$$

Nous utiliserons la notation glouton(V) pour désigner une politique gloutonne par rapport à V. Si π est une politique gloutonne par rapport à V, on a alors $TV = T_\pi V$. Si l'on connaît la fonction de valeur optimale, alors on en déduit une politique optimale π^* en sélectionnant une politique gloutonne par rapport à V^*.

1.1.3 Fonctions de valeur Q

Nous avons introduit les équations de Bellman et les notations dans le cas de fonctions de valeur définies sur l'espace d'états. Cependant, il est également possible d'utiliser des fonctions de valeur définies sur les couples états-actions. Cela peut être intéressant en particulier pour pouvoir calculer une politique gloutonne même lorsque le modèle du PDM n'est pas connu. Pour une politique π, une telle fonction de valeur est notée $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, où $Q^\pi(s,a)$ désigne l'espérance du cumul des récompenses que l'on peut obtenir à partir de l'état s, en effectuant l'action a et en suivant la politique π ensuite :

$$\forall (s,a) \in \mathcal{S} \times \mathcal{A} \qquad Q^\pi(s,a) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \,\middle|\, s_0 = s, a_0 = a, a_t = \pi(s_t) \text{ pour } t \geq 1 \right].$$

Une particularité des fonctions de valeur Q est que, contrairement au cas des fonctions de valeur V (voir l'équation (1.4)), calculer une politique gloutonne π par rapport à une fonction de valeur Q est immédiat :

$$\forall s \in \mathcal{S} \qquad \pi(s) \in \arg\max_{a \in \mathcal{A}} Q(s,a).$$

La connaissance du modèle du PDM (la fonction de transition P et la fonction de récompense R) n'est pas nécessaire pour calculer la politique gloutonne. Nous utiliserons également la notation glouton(Q) pour désigner une politique gloutonne par rapport à Q.

Dans le cas de fonctions de valeur Q, les équations de Bellman s'écrivent de façon légèrement différente. Pour une politique π, la fonction de valeur Q^π vérifie

$$\forall (s,a) \in \mathcal{S} \times \mathcal{A} \qquad Q^\pi(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s,a,s') Q^\pi(s', \pi(s')),$$

ce qui donne en notation condensée

$$Q^\pi = R + \gamma P_\pi Q^\pi$$

où Q^π est la fonction de valeur sous forme vectorielle :

$$Q^\pi = \begin{pmatrix} Q^\pi(s_1,a_1) \\ Q^\pi(s_1,a_2) \\ \vdots \\ Q^\pi(s_{|\mathcal{S}|},a_{|\mathcal{A}|}) \end{pmatrix},$$

R est le vecteur des récompenses de chaque couple état-action :

$$R = \begin{pmatrix} R(s_1,a_1) \\ R(s_1,a_2) \\ \vdots \\ R(s_{|\mathcal{S}|},a_{|\mathcal{A}|}) \end{pmatrix}$$

et P_π est la matrice de transition de la chaîne de Markov induite par le choix d'une action donnée suivie de la politique π ensuite : $P_\pi((s,a),(s',a')) = P(s,a,s')\pi(s',a')$. On note que R et P_π sont différents par rapport au cas des fonctions de valeur V. Nous conservons la même notation P_π pour la matrice de transition. Le vecteur des récompenses immédiates quant à lui ne dépend plus de la politique : nous le noterons donc R au lieu de R_π.

L'opérateur de Bellman T_π est défini pour tout vecteur Q par

$$T_\pi Q = R + \gamma P_\pi Q.$$

L'équation d'optimalité de Bellman se réécrit quant à elle de la manière suivante :

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s,a,s') \max_{a'} Q^*(s',a')$$

et l'opérateur d'optimalité T est donné pour tout vecteur Q par

$$[TQ](s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s,a,s') \max_{a'} Q(s',a').$$

Bertsekas et Tsitsiklis (1996, pages 245-246) ont montré qu'il existe une équivalence entre les fonctions de valeur V et les fonctions de valeur Q. Plus précisément, pour un PDM donné $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, toute fonction de valeur Q peut être vue comme étant seulement définie sur l'espace d'états si l'on considère un PDM auxiliaire. Dans ce PDM auxiliaire noté $< \mathcal{S}', \mathcal{A}', T', R', \gamma' >$, l'espace d'états S' est défini par $\mathcal{S}' = \mathcal{S} \cup (\mathcal{S} \times \mathcal{A})$. Dans un état s, l'action a mène de façon déterministe dans l'état (s,a). Puis une transition est effectuée selon T vers un état s' et cela donne lieu à une récompense selon R. Les états visités et les récompenses obtenues par une politique π dans ce PDM auxiliaire sont les mêmes que dans le PDM initial, en choisissant γ' de façon adaptée (c'est-à-dire $\gamma' = \sqrt{\gamma}$).

Sauf indication contraire, les algorithmes présentés dans ce mémoire concernent les fonctions de valeur V mais peuvent également s'appliquer aux fonctions de valeur Q. Nous utiliserons les fonctions de valeur Q uniquement dans le cas d'algorithmes nécessitant spécifiquement leur usage, par exemple lorsque l'on souhaitera calculer une politique gloutonne sans disposer du modèle du PDM.

1.2 Algorithmes fondamentaux

Nous présentons maintenant quelques algorithmes fondamentaux du contrôle optimal stochastique. Ces algorithmes permettent de calculer la fonction de valeur optimale et donc une politique optimale.

1.2.1 Value Iteration

Value Iteration (Puterman, 1994), issu de la programmation dynamique, est l'un des algorithmes standards des PDM. Il est décrit dans sa forme la plus usuelle dans l'algorithme 1.

Algorithme 1 Value Iteration (forme usuelle)

$k \leftarrow 0, V_0 \leftarrow$ initialisation arbitraire
répéter
  $V_{k+1} \leftarrow TV_k$
  $k \leftarrow k+1$
jusqu'à $\|V_k - V_{k-1}\|_\infty < \epsilon$

À chaque itération, on applique l'opérateur d'optimalité de Bellman T présenté plus haut. Comme cet opérateur est contractant et que son unique point fixe est la fonction de valeur optimale V^*, l'algorithme converge asymptotiquement vers la valeur optimale. Cependant, en pratique, il n'est pas garanti que la valeur optimale soit atteinte en un nombre d'itérations fini. On stoppe donc l'algorithme lorsque la distance entre deux valeurs successives devient inférieure à un certain seuil ϵ. On a alors une garantie sur la distance restante par rapport à la fonction de valeur optimale V^* (Bertsekas et Tsitsiklis, 1996) :

$$\|V^* - V_k\|_\infty \leq \frac{\gamma}{1-\gamma}\epsilon.$$

La performance de la politique gloutonne π_{k+1} par rapport à la valeur courante V_k vérifie quant à elle

$$\|V^* - V^{\pi_k}\|_\infty \leq \frac{2\gamma}{1-\gamma}\epsilon.$$

Il est possible d'écrire une forme alternative de l'algorithme Value Iteration. Cette forme alternative, donnée dans l'algorithme 2, est équivalente à la forme usuelle de l'algorithme 1 et nous permettra de mieux mettre en évidence le lien avec les algorithmes étudiés plus loin. L'idée est d'exprimer explicitement la politique gloutonne par rapport à V_k en remarquant que, dans l'algorithme 1, cette politique gloutonne est implicitement calculée lorsque l'on applique l'opérateur T.

Algorithme 2 Value Iteration (forme alternative équivalente)

$k \leftarrow 0, V_0 \leftarrow$ initialisation arbitraire
répéter
  $\pi_{k+1} \leftarrow \text{glouton}(V_k)$
  $V_{k+1} \leftarrow T_{\pi_{k+1}} V_k$
  $k \leftarrow k+1$
jusqu'à $\|V_k - V_{k-1}\|_\infty < \epsilon$

Comme π_{k+1} est la politique gloutonne par rapport à V_k, on a $T_{\pi_{k+1}} V_k = TV_k$. Ainsi, la valeur $V_{k+1} = T_{\pi_{k+1}} V_k$ calculée à chaque itération est bien la même que dans l'algorithme 1. Dans la suite de ce mémoire, nous considérerons toujours la forme de l'algorithme 2, plus proche de la démarche des autres algorithmes présentés.

1.2.2 Policy Iteration

Avec l'algorithme Policy Iteration (Puterman, 1994), la politique π_{k+1} est choisie comme la politique gloutonne sur les valeurs de V_k, puis V_{k+1} est calculée comme la valeur de la politique π_{k+1} (algorithme 3).

Algorithme 3 Policy Iteration (forme générale)

$k \leftarrow 0, V_0 \leftarrow$ initialisation arbitraire
répéter
  $\pi_{k+1} \leftarrow \text{glouton}(V_k)$
  $V_{k+1} \leftarrow V^{\pi_{k+1}}$
  $k \leftarrow k+1$
jusqu'à $\pi_k = \pi_{k-1}$

Pour calculer $V^{\pi_{k+1}}$, on peut résoudre directement l'équation de Bellman (équation 1.2), qui est un système linéaire :

$$\begin{aligned} V^{\pi_{k+1}} &= R_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} V^{\pi_{k+1}} \\ (I - \gamma P_{\pi_{k+1}}) V^{\pi_{k+1}} &= R_{\pi_{k+1}} \\ V^{\pi_{k+1}} &= (I - \gamma P_{\pi_{k+1}})^{-1} R_{\pi_{k+1}} \end{aligned}$$

Cependant, inverser la matrice $I - \gamma P_{\pi_{k+1}}$, qui est de taille $\mathcal{S} \times \mathcal{S}$, n'est possible en pratique que si le nombre d'états n'est pas trop élevé.

Une autre possibilité, donnée dans l'algorithme 4, est d'appliquer successivement l'opérateur $T_{\pi_{k+1}}$ jusqu'à atteindre son point fixe qui est la valeur de la politique π_{k+1}.

Algorithme 4 Policy Iteration (avec évaluation par itérations successives)

$k \leftarrow 0, V_0 \leftarrow$ initialisation arbitraire
répéter
  $\pi_{k+1} \leftarrow \text{glouton}(V_k)$
  $V_{k+1} \leftarrow T^{\infty}_{\pi_{k+1}} V_k$
  $k \leftarrow k+1$
jusqu'à $\pi_k = \pi_{k-1}$

La phase d'évaluation reste plus coûteuse en général que celle de Value Iteration puisqu'il faut calculer la valeur de la politique courante à chaque itération. En contrepartie, Policy Iteration nécessite en général moins d'itérations pour converger (Bertsekas et Tsitsiklis, 1996). Policy Iteration offre par ailleurs une garantie de convergence vers une politique optimale en un nombre fini d'itérations.

1.2.3 Modified Policy Iteration

Une technique intermédiaire entre Value Iteration et Policy Iteration consiste à appliquer à chaque itération l'opérateur de Bellman un nombre déterminé de fois m (algorithme 5). Ainsi, on ne calcule pas entièrement la valeur de la politique courante (contrairement à Policy Iteration), mais on peut s'en approcher plus rapidement qu'avec Value Iteration. Cette méthode est intitulée Modified Policy Iteration (Puterman, 1994).

Algorithme 5 Modified Policy Iteration

$m \in \mathbb{N}$
$k \leftarrow 0, V_0 \leftarrow$ initialisation arbitraire
répéter
  $\pi_{k+1} \leftarrow \text{glouton}(V_k)$
  $V_{k+1} \leftarrow T^{m}_{\pi_{k+1}} V_k$
  $k \leftarrow k+1$
jusqu'à $\|V_k - V_{k-1}\|_\infty < \epsilon$

Lorsque $m = 1$, on retrouve Value Iteration, et lorsque $m \to \infty$, on retrouve Policy Iteration. Il est établi que Modified Policy Iteration converge asymptotiquement vers la valeur optimale (Puterman, 1994).

1.2.4 λ-Policy Iteration

λ-Policy Iteration (λPI), introduit par Bertsekas et Ioffe (1996), propose une autre manière de généraliser Value Iteration et Policy Iteration. Comme dans les algorithmes précédents, la nouvelle politique π_{k+1} est choisie comme la politique gloutonne par rapport à V_k, puis on calcule une nouvelle fonction de valeur V_{k+1}. Un paramètre $\lambda \in [0,1]$ spécifie si la mise à jour de la fonction de valeur est plus proche de Policy Iteration ($\lambda = 1$) ou de Value Iteration ($\lambda = 0$). λ correspond à la taille du pas effectué en direction de $V^{\pi_{k+1}}$. Les auteurs de l'algorithme ont introduit un opérateur noté M_k et défini à l'itération k pour tout vecteur V par

$$M_k V = (1-\lambda) T_{\pi_{k+1}} V_k + \lambda T_{\pi_{k+1}} V. \tag{1.5}$$

Ils ont établi que l'opérateur M_k est contractant de facteur $\gamma\lambda$ pour la norme infinie. L'algorithme λPI calcule son point fixe en effectuant des applications successives de M_k (voir algorithme 6).

Algorithme 6 λ-Policy Iteration

$\lambda \in [0,1]$
$k \leftarrow 0, V_0 \leftarrow$ initialisation arbitraire
répéter
  $\pi_{k+1} \leftarrow \text{glouton}(V_k)$
  $V_{k+1} \leftarrow M_k^{\infty} V_k$
  $k \leftarrow k+1$
jusqu'à $\|V_k - V_{k-1}\|_\infty < \epsilon$

Intuitivement, cet opérateur M_k peut être vu comme une version **amortie** de l'opérateur de Bellman $T_{\pi_{k+1}}$: lorsque M_k est appliqué plusieurs fois de suite à un vecteur V, cela revient à construire une fonction de valeur en partie dans la direction de $T_{\pi_{k+1}} V_k$ (avec un poids $1-\lambda$), et en partie dans la direction de $V^{\pi_{k+1}}$ (avec un poids λ). Lorsque $\lambda = 1$, on a $M_k = T_{\pi_{k+1}}$ et l'algorithme se ramène exactement à Policy Iteration. Plus λ est grand, et plus le vecteur V_{k+1} calculé s'approche de $V^{\pi_{k+1}}$. A l'inverse, lorsque $\lambda = 0$, on a $V_{k+1} = T_{\pi_{k+1}} V_k$, ce qui correspond à Value Iteration. Plus λ est petit, plus on se contente d'une évaluation incomplète de la fonction de valeur avant de changer de politique.

Convergence de λPI

λPI converge vers la fonction de valeur optimale pour tout $\lambda \in [0,1]$ (Bertsekas et Ioffe, 1996). La vitesse de convergence asymptotique a été caractérisée analytiquement par ses auteurs. Nous rappelons ici ce résultat :

Proposition 1 (Convergence de λPI (Bertsekas et Ioffe, 1996))
Soit (V_k, π_k) la séquence de fonctions de valeurs et de politiques générées par λPI. On a alors :

$$\lim_{k \to +\infty} V_k = V^*.$$

De plus, pour tout k plus grand qu'un certain index $\bar{k}$,

$$\|V^* - V_{k+1}\|_\infty \leq \frac{\gamma(1-\lambda)}{1-\lambda\gamma} \|V^* - V_k\|_\infty.$$

$\bar{k}$ est l'itération à partir de laquelle une politique optimale est obtenue. On voit ici que le facteur $\beta = \frac{\gamma(1-\lambda)}{1-\lambda\gamma}$ est compris entre 0 (lorsque $\lambda = 1$) et γ (lorsque $\lambda = 0$). Lorsque la politique est optimale, la convergence asymptotique de la fonction de valeur est donc plus rapide pour les valeurs de λ proches de 1. Les petites valeurs de λ introduisent ainsi un biais, dû au fait que l'on ne calcule plus la fonction de valeur de la politique courante, mais que l'on se contente de s'en approcher.

L'opérateur d'évaluation incomplète T_λ

λPI étant un algorithme moins connu de la littérature, nous détaillons quelques unes de ses propriétés. Notons T_λ l'opérateur défini pour la politique en cours d'évaluation (π_{k+1}) et pour tout vecteur V par

$$T_\lambda V = (1-\lambda)\left(\sum_{i=1}^{\infty}\lambda^{i-1}T_{\pi_{k+1}}^i V\right). \tag{1.6}$$

L'opérateur T_λ calcule ainsi une moyenne géométrique de termes identiques à ceux de Modified Policy Iteration. Bertsekas et Ioffe (1996) ont montré qu'à l'itération k de λPI, on a

$$V_{k+1} = T_\lambda V_k.$$

Autrement dit, cet opérateur T_λ calcule la phase d'évaluation (incomplète si $\lambda < 1$) de λPI. On a en outre, pour tout $m \in \mathbb{N}^*$,

$$M_k^m V = (1-\lambda)\left(\sum_{i=1}^{m}\lambda^{i-1}T_{\pi_{k+1}}^i V_k\right) + \lambda^m T_{\pi_{k+1}}^m V. \tag{1.7}$$

L'équation (1.7) peut être vérifiée par induction. Pour $m = 1$, il s'agit de la définition de l'opérateur M_k (équation (1.5)). Pour $m > 1$, en développant l'expression $M_k^{m+1}V = M_k(M_k^m V)$, on obtient facilement l'équation (1.7) avec m remplacé par $m+1$. La relation $V_{k+1} = T_\lambda V_k$ est quant à elle obtenue à partir de l'égalité $V_{k+1} = M_k^\infty V_k$ et en prenant $m \to \infty$ dans l'équation (1.7).

Les auteurs de λPI (Bertsekas et Ioffe, 1996) ont encore montré une autre écriture possible du calcul de T_λ. On a en effet

$$T_\lambda V_k = V_k + \Delta_k, \tag{1.8}$$

où Δ_k est un vecteur de taille $\mathcal{S}$ défini par

$$\forall s \in \mathcal{S} \qquad \Delta_k(s) = E\left[\sum_{t=0}^{\infty}(\lambda\gamma)^t(r_{t+1} + \gamma V_k(s_{t+1}) - V_k(s_t))\,\middle|\, s_0 = s, a_t = \pi_{k+1}(s_t)\right].$$

On peut le vérifier en développant l'équation de point fixe de M_k :

$$\begin{aligned}
V_{k+1} &= M_k V_{k+1} \\
&= (1-\lambda)T_{\pi_{k+1}}V_k + \lambda T_{\pi_{k+1}}V_{k+1} \\
&= R_\pi + (1-\lambda)\gamma P_{\pi_{k+1}}V_k + \lambda\gamma P_{\pi_{k+1}}V_{k+1} \\
&= (I - \lambda\gamma P_{\pi_{k+1}})^{-1}[R_\pi + (1-\lambda)\gamma P_{\pi_{k+1}}V_k] \\
&= (I - \lambda\gamma P_{\pi_{k+1}})^{-1}[R_\pi + \gamma P_{\pi_{k+1}}V_k - V_k + V_k - \lambda\gamma P_{\pi_{k+1}}V_k] \\
&= (I - \lambda\gamma P_{\pi_{k+1}})^{-1}[R_\pi + \gamma P_{\pi_{k+1}}V_k - V_k + (I - \lambda\gamma P_{\pi_{k+1}})V_k] \\
&= V_k + (I - \lambda\gamma P_{\pi_{k+1}})^{-1}[R_\pi + \gamma P_{\pi_{k+1}}V_k - V_k] \\
&= V_k + \sum_{t=0}^{\infty}(\lambda\gamma P_{\pi_{k+1}})^t[R_\pi + \gamma P_{\pi_{k+1}}V_k - V_k] \\
\forall s \in \mathcal{S} \qquad V_{k+1}(s) &= V_k(s) + E\left[\sum_{t=0}^{\infty}(\lambda\gamma)^t(r_{t+1} + \gamma V_k(s_{t+1}) - V_k(s_t))\,\middle|\, s_0 = s, a_t = \pi_{k+1}(s_t)\right].
\end{aligned}$$

Cette expression sous forme d'espérance sera exploitée dans le chapitre 3 où nous présentons des méthodes pour estimer $T_\lambda V_k$ dans le cas approximatif.

En résumé, le vecteur $V_{k+1} = T_\lambda V_k$ calculé à chaque itération de λPI peut s'écrire de trois manières différentes :

$$\begin{aligned}
T_\lambda V_k &= (1-\lambda)\left(\sum_{i=1}^{\infty}\lambda^{i-1}T_{\pi_{k+1}}^i V_k\right) \\
&= M_k^\infty V_k \\
&= V_k + \Delta_k.
\end{aligned}$$

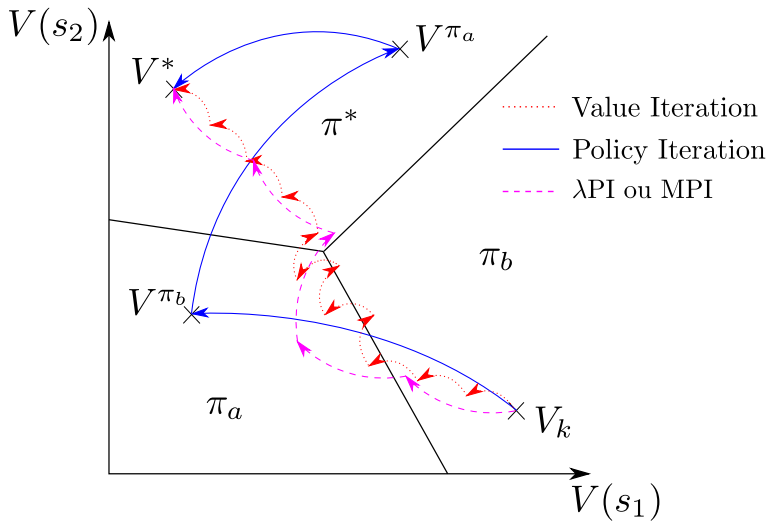


FIGURE 1.2 – **Vue intuitive de la notion d'optimisme dans la partition de l'espace des fonctions de valeur selon leur politique gloutonne :** D'après Bertsekas et Tsitsiklis (1996), on peut décomposer l'espace des fonctions de valeur en un ensemble de polyèdres, où chaque polyèdre correspond à une région où une politique est gloutonne. On suppose ici que l'espace d'états $\mathcal{S}$ ne contient que deux états s_1 et s_2 : l'espace des fonctions de valeur est ainsi un plan. Policy Iteration calcule un pas d'une seule étape directement vers $V^{\pi_{k+1}}$ tandis que Value Iteration réalise plusieurs petits pas en direction de $V^{\pi_{k+1}}$. Modified Policy Iteration (MPI) et λPI sont intermédiaires : ils réalisent une étape en direction de $V^{\pi_{k+1}}$, dont la longueur est paramétrable par m et λ respectivement. Plus la longueur des pas est petite, plus l'algorithme est dit optimiste.

1.2.5 La notion d'optimisme

Lorsqu'un algorithme change de politique alors que la fonction de valeur de la politique n'est pas encore atteinte, nous le qualifions dans ce mémoire d'**algorithme optimiste**, dans la mesure où il n'attend pas que l'évaluation soit complète avant de construire la nouvelle politique. Avec Policy Iteration, chaque itération fait un grand pas qui aboutit directement à $V^{\pi_{k+1}}$, valeur de la politique gloutonne, et il n'y a pas d'optimisme étant donné que la valeur est calculée entièrement. Avec Value Iteration, chaque itération fait un petit pas en direction de $V^{\pi_{k+1}}$: l'optimisme est ici maximal (l'évaluation est très incomplète). Modified Policy Iteration et λPI sont intermédiaires : l'optimisme est réglé par m et λ respectivement. Tous ces algorithmes peuvent ainsi être vus comme des variations plus ou moins optimistes de Policy Iteration.

La taille des pas effectués par ces algorithmes, ainsi que la trajectoire des fonctions de valeur successives, est illustrée sur la figure 1.2, qui représente une partition de l'espace des fonctions de valeur selon leur politique gloutonne à la manière de Bertsekas et Tsitsiklis (1996, page 227). L'espace des fonctions de valeur est découpé en plusieurs régions, où chaque région est un polyèdre qui correspond à un ensemble de fonctions de valeur ayant la même politique gloutonne.

Remarque

On peut vérifier analytiquement que ces régions sont bien des polyèdres. En effet, pour une politique π, la région correspondante notée G_π est définie par

$$\begin{aligned} G_\pi &= \{V \mid \pi \text{ est gloutonne par rapport à } V\} \\ &= \{V \mid T_\pi V = TV\} \end{aligned}$$

En développant les définitions de T_π et T, la condition $T_\pi V = TV$ devient

$$\begin{aligned} \forall s \in \mathcal{S} \qquad & R(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s,\pi(s),s')V(s') &&= \max_{a \in \mathcal{S}} \Big(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s,a,s')V(s') \Big) \\ \forall s \in \mathcal{S}, \forall a \in \mathcal{A} \qquad & R(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s,\pi(s),s')V(s') &&\geq R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s,a,s')V(s'). \end{aligned} \tag{1.9}$$

Ainsi, G_π est l'ensemble des fonctions de valeur V caractérisées par le système linéaire d'inégalités (1.9). On en déduit que G_π est un polyèdre dans l'espace des fonctions de valeur.

Conclusion

Nous avons présenté dans ce chapitre les algorithmes classiques du contrôle optimal stochastique Value Iteration, Policy Iteration et Modified Policy Iteration, ainsi que λPI (Bertsekas et Ioffe, 1996). Nous avons souligné le fait que tous ces algorithmes sont des formes d'itération sur les politiques qui diffèrent par la taille du pas effectué en direction de la valeur de la politique courante à chaque itération. Dans le chapitre 2, nous proposons une vision unifiée des méthodes d'itération sur les politiques avec optimisme réglable.

Chapitre 2

Une vision unifiée

Nous avons présenté plusieurs algorithmes du contrôle optimal stochastique : Value Iteration, Policy Iteration, Modified Policy Iteration et λ-Policy Iteration. Tous ces algorithmes calculent à chaque itération une politique gloutonne par rapport à la fonction de valeur courante, puis diffèrent par leur manière de calculer la fonction de valeur suivante. En observant la forme de la nouvelle fonction de valeur V_{k+1} (algorithmes 2, 4, 5 et 6), on constate qu'à chaque fois, celle-ci est une certaine combinaison linéaire de termes de la forme $T^i_{\pi_{k+1}} V_k$. En effet,

- dans Value Iteration, $V_{k+1} \leftarrow T_{\pi_{k+1}} V_k$;
- dans Policy Iteration, $V_{k+1} \leftarrow T^{\infty}_{\pi_{k+1}} V_k$;
- dans Modified Policy Iteration, $V_{k+1} \leftarrow T^{m}_{\pi_{k+1}} V_k$;
- dans λ-Policy Iteration, $V_{k+1} \leftarrow (1-\lambda) \sum_{i=1}^{\infty} \lambda^{i-1} T^i_{\pi_{k+1}} V_k$.

La seule différence entre ces algorithmes est la forme des coefficients de cette combinaison linéaire. Nous allons donc proposer une vision unifiée qui permet d'exprimer de façon générale toutes ces formes d'itération sur les politiques.

2.1 Unified Policy Iteration

En considérant une suite de coefficients positifis $\lambda_1, \ldots, \lambda_n, \ldots$ dont la somme est égale à 1, cette nouvelle méthode s'exprime comme décrit dans l'algorithme 7.

Algorithme 7 Unified Policy Iteration

$\lambda_i \in \mathbb{R}^+, \sum_{i=1}^{\infty} \lambda_i = 1.$
$k \leftarrow 0, V_0 \leftarrow$ initialisation arbitraire
répéter
  $\pi_{k+1} \leftarrow \text{glouton}(V_k)$
  $V_{k+1} \leftarrow \sum_{i=1}^{\infty} \lambda_i T^i_{\pi_{k+1}} V_k$
  $k \leftarrow k+1$
jusqu'à $\|V_k - V_{k-1}\|_\infty < \epsilon$

Le choix des coefficients λ_i permet de régler l'optimisme de l'algorithme. Le tableau 2.1 donne la valeur de ces coefficients pour chacun des algorithmes évoqués précédemment et unifiés sous cette forme.

Algorithme	**Mise à jour de la fonction de valeur**	**Coefficients**
Value Iteration	$V_{k+1} \leftarrow T_{\pi_{k+1}} V_k$	$\lambda_1 = 1$, $\lambda_i = 0$ pour $i \neq 1$
Policy Iteration	$V_{k+1} \leftarrow T^{\infty}_{\pi_{k+1}} V_k$	$\lambda_\infty = 1$, $\lambda_i = 0$ pour $i \neq \infty$
Modified Policy Iteration	$V_{k+1} \leftarrow T^{m}_{\pi_{k+1}} V_k$	$\lambda_m = 1$, $\lambda_i = 0$ pour $i \neq m$
λ-Policy Iteration	$V_{k+1} \leftarrow (1-\lambda) \sum_{i=1}^{\infty} \lambda^{i-1} T^i_{\pi_{k+1}} V_k$	$\lambda_i = (1-\lambda)\lambda^{i-1}$

TABLE 2.1 – Choix des coefficients λ_i dans les algorithmes classiques du contrôle optimal stochastique.

2.2 Résultat de convergence

Nous venons de proposer une version unifiée des algorithmes de type itération sur les politiques avec optimisme réglable. La question naturelle est de connaître les propriétés de convergence de cet algorithme général. Nous montrons dans cette section que Unified Policy Iteration converge vers la fonction de valeur optimale pour toute suite de coefficients positifs λ_i dont la somme est égale à 1, et nous établissons des bornes sur la vitesse de convergence. La proposition 2, qui énonce ce résultat, étend la proposition 1 (le cas de λPI) au cas général de Unified Policy Iteration et y ajoute une borne sur la vitesse de convergence non asymptotique.

Proposition 2 (Convergence de Unified Policy Iteration)
Soit $(\lambda_i)_{i\geq 1}$ une suite de coefficients réels positifs tels que $\sum_{i=1}^{\infty} \lambda_i = 1$. Soit (V_k, π_k) la séquence de fonctions de valeurs et de politiques générées par Unified Policy Iteration. On a alors :

$$\lim_{k\to+\infty} V_k = V^*.$$

De plus, pour tout k plus grand qu'un certain index $\bar{k}$,

$$\|V_{k+1} - V^*\|_\infty \leq \beta \|V_k - V^*\|_\infty, \text{ avec } \beta = \sum_{i=1}^{\infty} \lambda_i \gamma^i \in [0, \gamma].$$

Enfin, si V_0 est tel que $TV_0 \geq V_0$, alors on a pour tout k,

$$\|V_{k+1} - V^*\|_\infty \leq \gamma \|V_k - V^*\|_\infty.$$

Preuve

Nous nous inspirons ici de la preuve de convergence de λPI, dans Bertsekas et Tsitsiklis (1996, page 46) (ce résultat est rappelé dans la proposition 1). Supposons d'abord que $TV_0 \geq V_0$. Nous allons montrer par induction que pour tout k, on a

$$V^* \geq TV_{k+1} \geq V_{k+1} \geq TV_k \geq V_k. \tag{2.1}$$

L'opérateur $T_{\pi_{k+1}}$ est monotone (c'est-à-dire $V \geq V' \Rightarrow T_{\pi_{k+1}} V \geq T_{\pi_{k+1}} V'$) donc si $TV_k \geq V_k$, en utilisant le fait que $T_{\pi_{k+1}} V_k = TV_k$, on a

$$\begin{array}{lccccc} \forall i \geq 1, & T_{\pi_{k+1}} T^i_{\pi_{k+1}} V_k & \geq & T^i_{\pi_{k+1}} V_k & \geq & T_{\pi_{k+1}} V_k = TV_k \\ \forall i \geq 1, & \lambda_i T_{\pi_{k+1}} T^i_{\pi_{k+1}} V_k & \geq & \lambda_i T^i_{\pi_{k+1}} V_k & \geq & \lambda_i T_{\pi_{k+1}} V_k. \end{array}$$

Donc, en prenant la somme pour toutes les valeurs de i,

$$\begin{array}{lccccc} \sum_{i=1}^{\infty} \lambda_i T_{\pi_{k+1}} T^i_{\pi_{k+1}} V_k & \geq & \sum_{i=1}^{\infty} \lambda_i T^i_{\pi_{k+1}} V_k & \geq & \sum_{i=1}^{\infty} \lambda_i T_{\pi_{k+1}} V_k \\ T_{\pi_{k+1}} \sum_{i=1}^{\infty} \lambda_i T^i_{\pi_{k+1}} V_k & \geq & \sum_{i=1}^{\infty} \lambda_i T^i_{\pi_{k+1}} V_k & \geq & \sum_{i=1}^{\infty} \lambda_i T_{\pi_{k+1}} V_k. \end{array}$$

Comme $\sum_{i=1}^{\infty} \lambda_i = 1$, on obtient

$$T_{\pi_{k+1}} V_{k+1} \quad \geq \quad V_{k+1} \quad \geq \quad T_{\pi_{k+1}} V_k.$$

Or, $TV_{k+1} \geq T_{\pi_{k+1}} V_{k+1}$ (d'après la définition de T) et $T_{\pi_{k+1}} V_k = TV_k$, donc on a finalement

$$TV_{k+1} \quad \geq \quad V_{k+1} \quad \geq \quad TV_k.$$

Comme l'opérateur T est monotone, on a pour tout $n \in \mathbb{N}^*$, $T^n V_{k+1} \geq TV_{k+1}$. En prenant la limite quand $n \to +\infty$, on obtient

$$V^* \geq TV_{k+1},$$

ce qui, combiné avec l'inégalité précédente, montre l'inégalité (2.1) sous l'hypothèse $TV_0 \geq V_0$. En faisant tendre k vers ∞, la séquence des V_k converge donc vers une limite que l'on note V_∞ et qui vérifie $V^* \geq V_\infty$. On déduit alors de l'inégalité (2.1) que

$$V_\infty \geq TV_\infty \geq V_\infty.$$

On a donc $V_\infty = TV_\infty$, ce qui signifie que V_∞ vérifie l'équation de Bellman. Ainsi, $V_\infty = V^*$.

Pour montrer la vitesse de convergence non asymptotique, nous utilisons l'inégalité (2.1) ainsi que la propriété de contraction de facteur γ de l'opérateur T :

$$\begin{aligned}
\|V_{k+1} - V^*\|_\infty &= \max_{s \in \mathcal{S}} |V^*(s) - V_{k+1}(s)| \\
&= \max_{s \in \mathcal{S}} (V^*(s) - V_{k+1}(s)) \qquad \text{car } V^* \geq V_{k+1} \\
&\leq \max_{s \in \mathcal{S}} (TV^*(s) - TV_k(s)) \qquad \text{car } V_{k+1} \geq TV_k \\
&= \max_{s \in \mathcal{S}} |TV^*(s) - TV_k(s)| \qquad \text{car } V^* \geq TV_k \\
&= \|TV_k - TV^*\|_\infty \\
&\leq \gamma \|V_k - V^*\|_\infty.
\end{aligned}$$

Nous nous plaçons maintenant dans le cas où l'on n'a pas $TV_0 \geq V_0$. Pour montrer la convergence vers V^*, on peut remplacer V_0 par un vecteur $\hat{V}_0 = V_0 - ce$, où $e = (1, \ldots, 1)$ et c est une constante réelle positive suffisamment grande pour que $T\hat{V}_0 \geq \hat{V}_0$; en effet, on peut voir que lorsque $c \geq \frac{1}{1-\gamma} \max_s (V_0(s) - TV_0(s))$, on a $ce \geq \frac{1}{1-\gamma}(V_0 - TV_0)$ et donc $TV_0 - \gamma ce \geq V_0 - ce$, ce qui équivaut à $T\hat{V}_0 \geq \hat{V}_0$. Considérons alors l'algorithme Unified Policy Iteration initialisé avec $(\hat{V}_0, \pi_0)$ et notons $(\hat{V}_k, \hat{\pi}_k)$ la séquence des valeurs et politiques générées. Nous allons montrer par induction que pour tout k, on a

$$\hat{V}_k - V_k = -\beta^k ce \quad \text{avec } \beta = \sum_{i=1}^{\infty} \lambda_i \gamma^i \text{ et } \hat{\pi}_k = \pi_k.$$

Pour $k = 0$, on a $\hat{V}_0 - V_0 = V_0 - ce - V_0 = -ce$ et $\hat{\pi}_0 = \pi_0$ par définition. Supposons que la propriété est vérifiée au rang k. D'abord, $\hat{\pi}_{k+1} = \pi_{k+1}$ car $\hat{V}_k$ et V_k sont égales à une constante additive près. Ensuite, on peut écrire

$$\begin{aligned}
\hat{V}_{k+1} - V_{k+1} &= \sum_{i=1}^{\infty} \lambda_i T^i_{\hat{\pi}_{k+1}} \hat{V}_k - \sum_{i=1}^{\infty} \lambda_i T^i_{\pi_{k+1}} V_k \\
&= \sum_{i=1}^{\infty} \lambda_i (T^i_{\hat{\pi}_{k+1}} \hat{V}_k - T^i_{\pi_{k+1}} V_k) \\
&= \sum_{i=1}^{\infty} \lambda_i \gamma^i (\hat{V}_k - V_k) \qquad \text{car } \hat{\pi}_{k+1} = \pi_{k+1} \\
&= \beta (\hat{V}_k - V_k) \\
&= -\beta^{k+1} ce,
\end{aligned}$$

ce qui montre la propriété.

Or, β est la moyenne (pondérée par des coefficients λ_i dont la somme est égale à 1) des termes $\gamma, \gamma^2, \ldots, \gamma^n, \ldots$, donc $\beta \in [0, \gamma]$. Par conséquent, on a $\hat{V}_k - V_k \to 0$. Comme nous avons montré que $\hat{V}_k \to V^*$, on a bien également $V_k \to V^*$.

Montrons enfin la vitesse de convergence asymptotique. Considérons l'index $\bar{k}$ tel que pour tout $k \geq \bar{k}$, π_{k+1} est une politique optimale si bien que $T_{\pi_{k+1}} V^* = TV^* = V^*$. Alors, en utilisant le fait que l'opérateur $T_{\pi_{k+1}}$ est contractant de facteur γ, on a pour tout $k \geq \bar{k}$,

$$\begin{aligned}
\|V_{k+1} - V^*\|_\infty &= \|(\sum_{i=1}^{\infty} \lambda_i T^i_{\pi_{k+1}} V_k) - V^*\|_\infty \\
&= \|\sum_{i=1}^{\infty} \lambda_i (T^i_{\pi_{k+1}} V_k - V^*)\|_\infty \\
&\leq \sum_{i=1}^{\infty} \lambda_i \|T^i_{\pi_{k+1}} V_k - V^*\|_\infty \\
&= \sum_{i=1}^{\infty} \lambda_i \|T^i_{\pi_{k+1}} V_k - T^i_{\pi_{k+1}} V^*\|_\infty \\
&\leq \sum_{i=1}^{\infty} \lambda_i \gamma^i \|V_k - V^*\|_\infty \\
&= \beta \|V_k - V^*\|_\infty. \quad \blacksquare
\end{aligned}$$

2.3 Illustration : Modified λ-Policy Iteration

La proposition 2 montre que tout algorithme qui peut s'exprimer sous la forme de Unified Policy Iteration converge vers la fonction de valeur optimale. Afin d'illustrer ce résultat, nous proposons et étudions ici un tel algorithme. Ce nouvel algorithme combine les idées de λPI (Bertsekas et Ioffe, 1996) et de Modified Policy Iteration (Puterman, 1994).

Rappelons que λPI (voir l'algorithme 6) s'approche de la valeur de la politique courante d'un certain pas dont la taille est réglable par un paramètre $\lambda \in [0, 1]$, et qu'il peut pour cela appliquer répétitivement l'opérateur M_k jusqu'à obtenir son point fixe. Pour rappel, l'opérateur M_k est défini à l'itération k de λPI pour tout vecteur V comme $M_k V = (1 - \lambda) T_{\pi_{k+1}} V_k + \lambda T_{\pi_{k+1}} V$. L'idée est de modifier λPI en se contenant d'appliquer l'opérateur M_k un nombre limité de fois. Cette modification vise à rendre la phase d'évaluation de la politique plus souple, en s'arrêtant sans attendre d'avoir convergé précisément vers le point fixe de l'opérateur M_k. Nous appelons cette méthode Modified λ-Policy Iteration, de manière analogue à Modified Policy Iteration (voir algorithme 5) qui repose sur la même idée : stopper l'évaluation après un nombre limité d'étapes.

Algorithme 8 Modified λ-Policy Iteration

$\lambda \in [0, 1], m \in \mathbb{N}*$
$k \leftarrow 0, V_0 \leftarrow$ initialisation arbitraire
répéter
  $\pi_{k+1} \leftarrow \text{glouton}(V_k)$
  $V_{k+1} \leftarrow M_k^m V_k$
  $k \leftarrow k + 1$
jusqu'à $\|V_k - V_{k-1}\|_\infty < \epsilon$

Modified λ-Policy Iteration prend donc deux paramètres : $\lambda \in [0, 1]$ provenant de λPI, et $m \in \mathbb{N}^*$ provenant de Modified Policy Iteration. La mise à jour de la fonction de valeur consiste à appliquer m fois l'opérateur M_k : $V_{k+1} \leftarrow M_k^m V_k$. En observant la forme de $M_k^m V_k$ sur l'équation (1.7) (page 27), on peut voir que Modified λ-Policy Iteration est bien une implémentation de Unified Policy Iteration. En

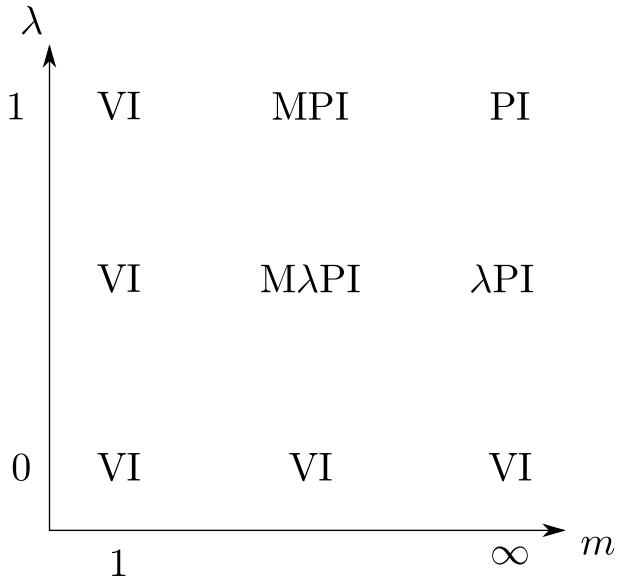


FIGURE 2.1 – Selon la valeur des paramètres λ et m, Modified λ-Policy Iteration (MλPI) généralise les algorithmes λ-Policy Iteration (λPI), Modified Policy Iteration (MPI), Policy Iteration (PI) et Value Iteration (VI).

effet, Modified λ-Policy Iteration consiste à prendre $\lambda_i = (1-\lambda)\lambda^{i-1}$ pour $i < m$, $\lambda_m = \lambda^m$ et $\lambda_i = 0$ pour $i > m$.

Notons enfin que Modified λ-Policy Iteration généralise à la fois λPI et Modified Policy Iteration. Plus précisément, comme la mise à jour de la fonction de valeur est $V_{k+1} \leftarrow M_k^m V_k$, on peut voir que :

- si $m \to \infty$, on obtient l'algorithme λ-PI,
- si $\lambda = 1$, on obtient l'algorithme Modified Policy Iteration car on a alors $M_k = T_{\pi_{k+1}}$,
- si les deux conditions précédentes sont réunies, on obtient l'algorithme Policy Iteration,
- si $m = 1$ ou si $\lambda = 0$, on obtient l'algorithme Value Iteration.

Ces généralisations sont récapitulées sur le schéma de la figure 2.1.

Expériences

Afin d'étudier empiriquement l'influence de λ et m, nous avons mené des expériences sur un problème de type navigation discrète. Un agent se déplace sur une grille en deux dimensions et se dirige dans les quatre directions principales jusqu'à atteindre un objectif. Certaines cases de la grille sont des murs et les décisions de l'agent peuvent être bruitées. Plus formellement, le PDM est défini de la manière suivante.

- L'espace d'états $\mathcal{S}$ est l'ensemble des cases de la grille n'étant pas des murs, auxquelles on ajoute un état terminal indiquant que l'objectif a été atteint.
- L'espace d'actions $\mathcal{A}$ est composé des cinq actions suivantes : Nord, Sud, Est, Ouest et l'action consistant à rester sur place.
- Notons $\mu \in [0,1]$ un terme de bruit appliqué au déplacement. La fonction de transition est, avec probabilité $1-\mu$, le déplacement correspondant l'action choisie, et avec probabilité μ, un déplacement aléatoire choisi uniformément parmi les quatre directions. Lorsque l'action choisie consiste à rester sur place, aucun bruit n'est appliqué. Si un déplacement mène à une case occupée par un mur, alors l'agent reste sur place.
- Enfin, la récompense est de -1 à chaque étape tant que l'état terminal n'est pas atteint, 0 une fois que l'état terminal est atteint, et une pénalité de -100 en cas de collision contre un mur.

Nous avons considéré plusieurs environnements avec différentes valeurs de γ et du bruit μ, sur lesquels nous avons exécuté l'algorithme Modified λ-Policy Iteration avec différentes valeurs de λ et de m afin de rechercher des PDM pour lesquels la meilleure valeur de λ serait différente de 1. Pour comparer

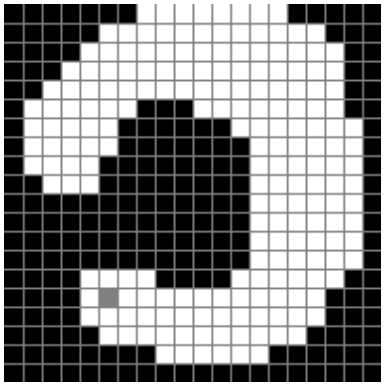


FIGURE 2.2 – L'environnement utilisé pour les expériences de la figure 2.3. Les cases noires représentent les murs et la case grise représente l'état terminal.

les exécutions en termes de rapidité, nous avons défini une mesure de performance destinée à compter le nombre d'opérations effectuées au cours de l'exécution d'un algorithme. Nous considérons qu'une application de l'opérateur de Bellman T_π correspond à une opération (cet opérateur nécessite de parcourir tous les états du PDM). Le calcul d'une politique gloutonne équivaut à $|\mathcal{A}|$ opérations (5 dans notre cas), car pour chaque action il faut parcourir tous les états. Enfin, calculer $V_{k+1} = M_k^m V_k$ nécessite $m+1$ opérations : une opération pour calculer le terme $(1-\lambda)T_{\pi_{k+1}}V_k$, qui ne change pas lorsque l'on applique M_k plusieurs fois de suite, et m opérations pour le second terme de M_k, qui demande une opération à chaque fois que M_k est appliqué.

La figure 2.3 représente, pour un environnement donné de taille 20×20, le nombre d'opérations qui ont été nécessaires pour faire converger Modified λ-Policy Iteration vers la fonction de valeur optimale en fonction de différentes valeurs de λ et de m. La convergence de la fonction de valeur n'étant qu'asymptotique, on arrête l'algorithme lorsque $\|V_k - V_{k-1}\|_\infty \leq \epsilon$, avec $\epsilon = 10^{-10}$. Les propriétés de ce PDM sont $\gamma = 0,999$ et $\mu = 0.4$. L'environnement utilisé pour ces expériences est illustré sur la figure 2.2.

Les paramètres pour lesquels le nombre d'opérations est minimal (387 opérations) sont $\lambda = 1$ et $m = 32$. Cette courbe est typique des expériences que nous avons lancées : en effet, sur toutes nos expériences, il apparaît que le nombre d'opérations est le plus faible lorsque m est limité (typiquement, $10 < m < 100$) et $\lambda = 1$, et ce pour toutes les formes d'environnements que nous avons testées. Choisir $\lambda < 1$ ne semble être intéressant que si m est grand.

Pour certains PDM cependant, la valeur optimale de λ qui a été trouvée s'est avérée être légèrement inférieure à 1 (entre $0,97$ et 1 selon les cas). La figure 2.4 représente des résultats plus précis pour l'un de ces PDM. Ses propriétés sont $\gamma = 0,998$ et un bruit est de $0,1$. Les meilleurs paramètres trouvés étaient $\lambda = 0,99$ et $m = 4$. Nous avons relancé des expériences sur ce PDM avec des valeurs plus fines de λ et m afin d'obtenir une courbe plus précise autour de ces valeurs (voir figure 2.4). On remarque en fait que les résultats sont peu concluants. Il semble qu'à cause de la structure du PDM, lorsque l'on modifie légèrement λ ou m, la trajectoire des politiques π_k peut changer et ainsi aboutir à une politique optimale parfois plus tôt que d'autres sans pour autant qu'un choix de ces paramètres se montre clairement avantageux. Dans ces conditions, ces résultats ne permettent donc pas de conclure qu'une valeur de λ différente de 1 puisse être significativement meilleure que $\lambda = 1$ en termes de vitesse de convergence lorsque m est fixé à une valeur donnant de bonnes performances.

Discussion

L'intérêt d'utiliser $\lambda < 1$ est d'accélérer chaque itération en évaluant de manière incomplète la politique courante π_{k+1}. En contrepartie, le nombre total d'itérations augmente puisque cette évaluation n'est plus complète. Or, dès lors que l'on fait diminuer m, c'est également ce qui se passe : on limite le nombre d'applications de l'opérateur M_k. Il apparaît donc que dans ce cas, il n'est plus utile d'avoir $\lambda < 1$ car cela ne fait que rendre plus incomplète l'évaluation de π_{k+1}. Utiliser $\lambda = 1$ ne pose alors plus de problème

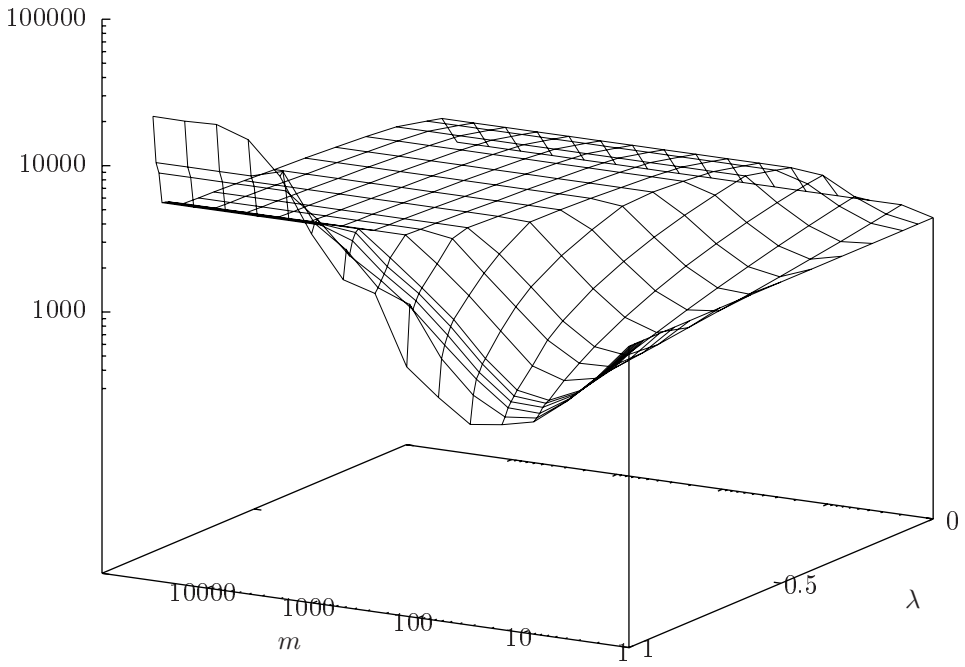


FIGURE 2.3 – Nombre d'opérations effectuées par l'algorithme Modified λ-Policy Iteration pour converger en fonction des paramètres λ et m, sur un problème où $\gamma = 0,999$ et le bruit est de $0,4$. Le minimum est atteint pour $\lambda = 1$ et $m = 32$, où 387 opérations ont été effectuées. Lorsque le nombre d'étapes m est en-dessous d'une certaine valeur, il n'est plus utile d'utiliser une valeur de λ autre que 1.

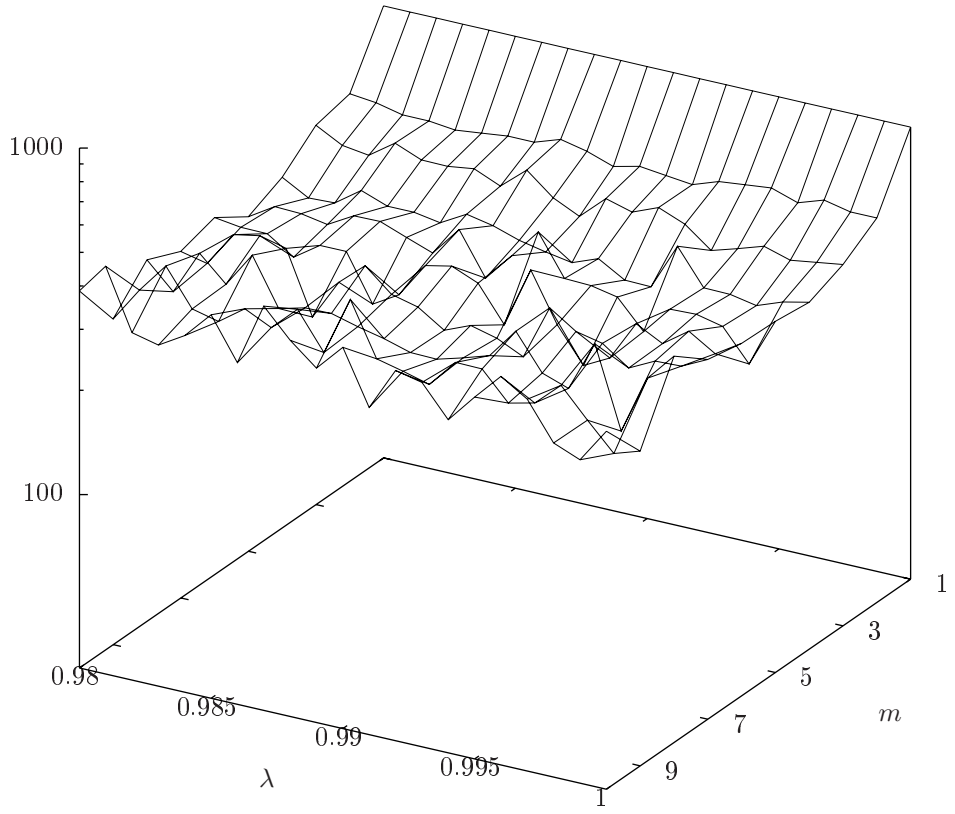


FIGURE 2.4 – Nombre d'opérations effectuées par l'algorithme Modified λ-Policy Iteration en fonction d'un champ restreint de paramètres λ et m ($\lambda \in [0,98;1]$ et $m \in [1;10]$), avec $\gamma = 0,998$ et un bruit de $0,1$. Bien que le minimum soit atteint pour $\lambda = 0,994$ et $m = 6$ (avec 223 opérations effectuées), les résultats semblent peu significatifs et ne permettent pas de conclure qu'une valeur de λ différente de 1 soit clairement meilleure.

étant donné que la durée des itérations est déjà limitée par le nombre d'opérations m.

Ces expériences montrent que dans le cas exact, l'intérêt d'utiliser le paramètre λ est donc limité. Lorsque $\lambda < 1$, le biais dû à l'évaluation optimiste dégrade la vitesse de convergence asymptotique et rien ne vient compenser ce biais. En pratique, c'est Modified Policy Iteration (algorithme 5), plus direct que λPI, qui induit la meilleure convergence dans le cas exact. Comme nous le verrons dans les prochains chapitres, c'est dans le cas approché que λ va révéler son utilité.

Conclusion

Dans ce chapitre, nous avons introduit Unified Policy Iteration, une écriture unifiée des algorithmes du contrôle optimal stochastique qui calculent une politique tels que ceux présentés dans le chapitre 1. Nous avons établi la convergence de cet algorithme unifié et donné des bornes sur la vitesse de convergence asymptotique et non asymptotique, en généralisant le résultat existant sur λPI (Bertsekas et Ioffe, 1996). Ce résultat a l'avantage de s'appliquer de façon générale à tout algorithme qui calcule une politique avec de l'optimisme, avec peu de contraintes sur la manière dont est mise à jour la fonction de valeur.

Afin d'illustrer Unified Policy Iteration par un exemple, nous avons proposé MλPI, un algorithme qui s'inscrit dans ce cadre et qui cumule les concepts de Modified Policy Iteration et de λPI. Notre étude empirique sur un PDM de type navigation discrète a montré que, lorsque le paramètre m de Modified Policy Iteration est bien réglé, il est peu utile d'employer le paramètre λ de λPI.

Cependant, nous nous sommes limités jusqu'ici au cas exact où la fonction de valeur est représentée de façon tabulaire, et où le modèle du PDM (la fonction de transition P et la fonction de récompenses R) est connu. Or, le véritable problème de l'apprentissage par renforcement consiste à calculer une politique sans connaître nécessairement le modèle du PDM, et même lorsque le modèle du PDM est connu, le grand nombre d'états dans les applications réelles empêche les algorithmes exacts d'être applicables directement. Pour traiter ces difficultés, les techniques d'apprentissage par renforcement s'appuient sur les méthodes du contrôle optimal stochastique que nous avons vues jusqu'ici et y ajoutent des techniques d'estimation et d'approximation de la fonction de valeur. Nous allons voir que λPI, dans le cas approché, possède des caractéristiques avantageuses notamment grâce aux propriétés de l'opérateur T_λ. Le fait d'employer $\lambda < 1$ permettra d'améliorer l'estimation de la fonction de valeur lorsque des échantillons sont utilisés.

Deuxième partie

Le cas approché

Chapitre 3

Apprentissage par renforcement avec approximation

Nous supposions jusqu'ici que les fonctions de valeur étaient représentées de manière exacte, c'est-à-dire sous une forme tabulaire où la valeur de chaque état est stockée explicitement. Cependant, si cela peut suffire pour traiter des problèmes où l'espace d'états est réduit, de nombreux problèmes réels nécessitent un nombre d'états beaucoup plus important. Ainsi, en général, le nombre d'états à stocker en mémoire et le temps nécessaire pour les parcourir individuellement ne permettent pas une résolution exacte en pratique. Il va donc falloir parvenir à généraliser à un grand nombre d'états l'expérience acquise sur un petit échantillon d'états visités, afin de construire une bonne approximation de la fonction de valeur. Autrement dit, il s'agit de recourir à des techniques qui combinent **l'approximation de fonctions** et **l'estimation à partir d'échantillons** .

Au lieu de représenter la fonction de valeur de façon tabulaire, il est usuel de l'approcher avec une fonction paramétrée. La fonction de valeur exacte V_k est ainsi remplacée par une représentation approchée $\widehat{V}_k = f(w_k)$, où w_k est un vecteur de paramètres. On commet alors une **erreur d'approximation** $\epsilon_k = \widehat{V}_k - V_k$. Il n'est plus nécessaire de stocker la valeur de chaque état, mais uniquement de stocker les paramètres w_k de cette représentation, ce qui demande en général beaucoup moins de ressources. Les techniques d'approximation cherchent à régler ces paramètres en minimisant l'erreur commise, de manière à ce que la fonction de valeur approximative soit la plus proche possible de la fonction de valeur ciblée. Pour cela, deux problèmes se posent : comment choisir l'architecture (la fonction f), et comment ajuster les paramètres w_k ? Nous abordons dans cette thèse essentiellement la seconde question, c'est-à-dire que pour une architecture donnée, nous nous intéressons à des méthodes permettant de déterminer les paramètres w_k.

Dans ce chapitre, nous introduisons l'apprentissage par renforcement avec approximation. Nous énonçons un résultat analytique qui garantit la performance des algorithmes de type itération sur les politiques optimiste dans le cas approché, puis nous nous intéressons à plusieurs algorithmes majeurs de l'état de l'art de l'approximation linéaire. Ces algorithmes sont le fondement de la contribution que nous présenterons dans le chapitre 4.

3.1 Borne de performance

Nous avons précédemment introduit Unified Policy Iteration, une écriture qui généralise les méthodes classiques de la progammation dynamique telles que Value Iteration, Policy Iteration et Modified Policy Iteration, mais aussi λ-Policy Iteration (Bertsekas et Ioffe, 1996). Unified Policy Iteration permet notamment d'introduire de l'optimisme dans l'évaluation de la fonction de valeur selon le choix des coefficients λ_i, c'est-à-dire d'évaluer la politique de façon incomplète. Nous avons montré que dans le cas exact, Unified Policy Iteration converge vers une politique optimale pour tout choix des coefficients λ_i (voir la proposition 2, page 32). Il est donc naturel de se demander s'il existe également dans le cas approché une garantie sur la performance des politiques générées par cet algorithme. Nous montrons dans cette

section qu'une telle garantie existe en effet. Le résultat énoncé dans le théorème 1 montre qu'il est raisonnable d'appliquer Unified Policy Iteration avec approximation, c'est-à-dire en commettant une erreur ϵ_k à l'itération k, pour tout choix des coefficients λ_i, à partir du moment où cette erreur d'approximation ϵ_k est bornée à chaque itération. De telles garanties sont déjà connues dans les cas particuliers de Value Iteration et Policy Iteration avec approximation (Bertsekas et Tsitsiklis, 1996), ainsi que dans le cas de λPI avec approximation (Scherrer, 2007), mais pas dans le cas général des algorithmes d'itération sur les politiques optimistes qui s'inscrivent dans le cadre de Unified Policy Iteration. Ainsi, ce théorème donne les premières garanties de performance pour les versions approximatives de Modified Policy Iteration (et de la même façon, pour les versions approximatives de l'algorithme Modified λ-Policy Iteration introduit au chapitre 2).

Dans Bertsekas et Tsitsiklis (1996, section 6.4, page 320), les auteurs écrivent ainsi *"This leaves us with a major theoretical question. Is there some variant of optimistic policy iteration that is guaranteed to generate policies whose performance is within $O(\epsilon/(1-\alpha))$ or even $O(\epsilon/(1-\alpha)^2)$ from the optimal ?"*. Le théorème que nous énonçons ici est la première borne de performance pour les algorithmes optimistes [3] d'itération sur les politiques approchés. Il s'agit d'un résultat abstrait qui est indépendant d'un type d'approximation de la fonction de valeur.

Théorème 1 (Borne sur la performance de Unified Policy Iteration approché)

Soit un ensemble de poids positifs $(\lambda_i)_{i\geq 1}$ tels que $\sum_{i=1}^{\infty} \lambda_i = 1$. Soit une initialisation quelconque $\widehat{V}_0$. Soit un algorithme itératif qui construit la suite $(\pi_k, \widehat{V}_k)_{k\geq 1}$ de la manière suivante :

$$\begin{aligned} \pi_{k+1} &\leftarrow \text{glouton}(\widehat{V}_k) \\ \widehat{V}_{k+1} &\leftarrow \sum_{i=1}^{\infty} \lambda_i (T_{\pi_{k+1}})^i \widehat{V}_k + \epsilon_{k+1}. \end{aligned}$$

ϵ_{k+1} représente l'erreur d'approximation commise en estimant V_{k+1}. Supposons qu'il existe une majoration uniforme ϵ de l'erreur : pour tout k, $\|\epsilon_k\|_\infty \leq \epsilon$. Alors

$$\limsup_{k\to\infty} \|V^* - V^{\pi_k}\|_\infty \leq \frac{2\gamma}{(1-\gamma)^2}\epsilon.$$

Idée générale de la preuve (Scherrer et Thiery, 2010)

La preuve de ce théorème, qui se trouve en annexe A, est significativement différente de celles qui ont été proposées (séparément) pour les versions approximatives de Value Iteration et Policy Iteration (Bertsekas et Tsitsiklis, 1996). Dans le cas de Policy Iteration, le raisonnement s'appuie sur la propriété de croissance des fonctions de valeur et, dans le cas de Value Iteration, il utilise des arguments liés aux contractions. Ces deux types d'arguments ne peuvent pas être utilisés dans le cadre général de ce théorème.

Nous donnons ici une explication intuitive sur la démarche de cette preuve. La performance de la politique courante π_k est mesurée par la distance entre sa valeur et la valeur de la politique optimale : $\|V^* - V^{\pi_k}\|_\infty$. L'idée est de décomposer cette distance en deux termes de la façon suivante :

$$\begin{aligned} \|V^* - V^{\pi_k}\|_\infty &= \max(V^* - V^{\pi_k}) \\ &= \max\Big(\underbrace{V^* - (\widehat{V}_k - \epsilon_k)}_{d_k} + \underbrace{(\widehat{V}_k - \epsilon_k) - V^{\pi_k}}_{s_k}\Big) \end{aligned}$$

Rappelons que $\widehat{V}_k - \epsilon_k = V_k$ est la fonction de valeur ciblée par l'algorithme à l'itération k (sans erreur). d_k représente la distance à l'optimal de la fonction de valeur ciblée V_k. s_k représente quant à lui le biais

3. La notion d'optimisme décrite par Bertsekas et Tsitsiklis (1996, chapitres 5.4 et 6.4), est analogue (quoiqu'un peu plus extrême) à celle qui est employée dans ce mémoire. Nous reprenons ce terme dans le sens où l'on change de politique avant d'avoir fini de calculer la valeur de la politique précédente.

introduit par l'optimisme, c'est-à-dire le fait que la fonction de valeur ciblée V_k n'est pas nécessairement la valeur de la politique V^{π_k}.

La démarche de la preuve consiste à calculer séparément des majorations de d_k et s_k. Il se trouve que ces majorations dépendent toutes deux d'une majoration de l'erreur de Bellman $\widehat{V}_k - T_{\pi_{k+1}}\widehat{V}_k$. La borne de performance est finalement obtenue en utilisant ces deux majorations.

3.2 Architecture d'approximation linéaire

Nous venons de présenter une borne de performances abstraite qui se place dans un cadre d'itération sur les politiques avec optimisme éventuel mais qui peut concerner tout type d'approximation de la fonction de valeur. Dans cette thèse, nous nous intéressons à un type d'architecture : **l'approximation linéaire** de la fonction de valeur. Même si les architectures linéaires ont une capacité d'approximation moindre que celle des méthodes non linéaires telles que les réseaux de neurones, elles sont plus simples à mettre en œuvre et à analyser et font l'objet de nombreux travaux (Bertsekas et Ioffe, 1996; Bradtke et Barto, 1996; Sutton et Barto, 1998; Boyan, 2002; Lagoudakis et Parr, 2003; Nedić et Bertsekas, 2003; Yu et Bertsekas, 2009; Munos, 2004). De plus, il a été montré que les architectures non linéaires peuvent diverger là où des garanties existent pour le cas linéaire (Tsitsiklis et Roy, 1997).

Nous présentons ici les notations spécifiques à l'approximation linéaire, et en particulier aux méthodes d'itération sur les politiques. À chaque itération k, on maintient à jour une politique π_k et une fonction de valeur approximative $\widehat{V}_k$. On considère une architecture d'approximation linéaire classique, où la fonction de valeur $\widehat{V}_k$ est représentée avec une combinaison linéaire de p fonctions de base :

$$\widehat{V}_k(s) = \sum_{i=1}^{p} w_k(i)\phi_i(s).$$

Les termes $\phi_i(s)$ sont p fonctions de base arbitraires et les $w_k(i)$ sont les p paramètres de l'architecture. Comme en général $p \ll |\mathcal{S}|$ lorsque l'espace d'états est grand, stocker une fonction de valeur ainsi représentée demande beaucoup moins d'espace qu'une représentation tabulaire. En notant $\phi(s)$ le vecteur de taille p dont les éléments sont les fonctions de base appliquées à l'état s, c'est-à-dire

$$\phi(s) = \begin{pmatrix} \phi_1(s) \\ \vdots \\ \phi_p(s) \end{pmatrix},$$

et Φ la matrice de taille $|\mathcal{S}| \times p$ composée de tous ces vecteurs, c'est-à-dire

$$\Phi = \begin{pmatrix} — & \phi(s_1)^\mathsf{T} & — \\ & \vdots & \\ — & \phi(s_{|\mathcal{S}|})^\mathsf{T} & — \end{pmatrix} = \begin{pmatrix} | & & | \\ \phi_1 & \cdots & \phi_p \\ | & & | \end{pmatrix},$$

$\widehat{V}_k$ peut être noté $\widehat{V}_k = \Phi w_k$, où w_k est le vecteur des paramètres $(w_k(1), \ldots, w_k(p))^\mathsf{T}$ caractérisant la fonction de valeur approximative $\widehat{V}_k$. On a en outre, pour tout état s, $\widehat{V}_k(s) = \phi(s)^\mathsf{T} w_k(s)$.

Dans un algorithme de type itération sur les politiques, π_{k+1} est choisie comme étant la politique gloutonne par rapport à $\widehat{V}_k$, puis on représente $\widehat{V}_{k+1}$ comme une nouvelle combinaison linéaire des fonctions de base, en calculant un nouveau vecteur de paramètres w_{k+1}.

Cas des fonctions de valeur Q

Dans le cas où l'on utilise des fonctions de valeur Q (définies sur l'espace des couples états-actions), les fonctions de base sont elles aussi définies sur les couples états-actions. Les notations de l'approximation linéaire s'étendent naturellement à ce cas de figure à quelques modifications près. La fonction de valeur est approchée par la combinaison linéaire suivante :

$$\widehat{Q}_k(s,a) = \sum_{i=1}^{p} w_k(i)\phi_i(s,a).$$

De manière similaire, on peut écrire $\widehat{Q}_k = \Phi w_k$, en notant $\phi(s,a)$ le vecteur de taille p dont les éléments sont les fonctions de base appliquées couple (s,a), c'est-à-dire

$$\phi(s,a) = \begin{pmatrix} \phi_1(s,a) \\ \vdots \\ \phi_p(s,a) \end{pmatrix}$$

et Φ la matrice de taille $|\mathcal{S}||\mathcal{A}| \times p$ composée de tous ces vecteurs, c'est-à-dire

$$\Phi = \begin{pmatrix} — & \phi(s_1,a_1)^\mathsf{T} & — \\ — & \phi(s_1,a_2)^\mathsf{T} & — \\ & \vdots & \\ — & \phi(s_{|\mathcal{S}|},a_{|\mathcal{A}|})^\mathsf{T} & — \end{pmatrix} = \begin{pmatrix} | & & | \\ \phi_1 & \cdots & \phi_p \\ | & & | \end{pmatrix}.$$

Les notations spécifiques à l'approximation linéaire de la fonction de valeur étant introduites, nous présentons maintenant quelques algorithmes standards d'évaluation de la politique ou d'itération sur les politiques avec approximation linéaire de la fonction de valeur.

3.3 Approximation linéaire du premier ordre

Les algorithmes d'approximation linéaire se déclinent en deux familles de méthodes d'approximation : les méthodes du premier ordre et celles du second ordre. Si les deux types de méthodes s'appuient sur des échantillons pour évaluer les politiques, elles diffèrent par leur manière d'exploiter les informations que contiennent ces derniers.

Les méthodes du premier ordre ont pour principe d'effectuer, pour chaque échantillon observé, une petite correction de la fonction de valeur approximative en direction de la fonction de valeur réelle. Cette correction se veut simple à calculer, la complexité du calcul étant de $O(p)$ où p est la dimension de l'architecture linéaire. On s'appuie pour cela sur les récompenses observées jusqu'à une certaine profondeur, puis au-delà de cette profondeur, sur l'estimation courante de la fonction de valeur. Nous mentionnons ici les deux algorithmes fondamentaux du premier ordre que sont TD(0) et TD(λ) (Sutton et Barto, 1998).

3.3.1 TD(0)

TD(0) avec approximation linéaire (Sutton et Barto, 1998) est un algorithme fondateur de la communauté de l'apprentissage par renforcement. Il a pour objectif d'estimer la valeur V^π d'une politique π donnée, à partir d'une trajectoire[4] d'échantillons générée avec π. Si l'on se place dans un schéma d'itération sur les politiques, la politique π à évaluer est π_{k+1}, gloutonne par rapport à la valeur précédente, et k désigne l'itération courante.

Idéalement, on souhaiterait que la fonction de valeur approximative $\widehat{V}$ vérifie l'équation de Bellman : $\widehat{V} = T_\pi \widehat{V}$. Comme $T_\pi \widehat{V}$ n'appartient en général pas à l'espace des fonctions de valeur approximatives, une possibilité est de le projeter sur cet espace. On va donc chercher à résoudre l'équation projetée $\widehat{V} = \Pi_{D_\pi} T_\pi \widehat{V}$, où Π_{D_π} est une projection orthogonale sur l'espace d'approximation.

Formellement, la matrice de projection orthogonale Π_{D_π} est définie par $\Pi_{D_\pi} = \Phi(\Phi^\mathsf{T} D_\pi \Phi)^{-1} \Phi^\mathsf{T} D_\pi$. D_π représente la matrice diagonale de taille $|\mathcal{S}|$ dont les termes sont les poids de la distribution stationnaire μ associée à la politique π. Π_{D_π} correspond à la projection orthogonale selon la norme quadratique pondérée par μ, notée $\|\cdot\|_{\mu,2}$ et définie pour tout vecteur V par

$$\|V\|_{\mu,2} = \sqrt{\sum_{s\in\mathcal{S}} \mu(s) V(s)^2}.$$

Le principe de projeter de $T_\pi \widehat{V}$ sur l'espace d'approximation est illustré sur la figure 3.1.

4. On peut étendre de façon assez immédiate au cas de plusieurs trajectoires TD(0) et TD(λ) (voir Sutton et Barto, 1998), de même que les algorithmes présentés plus loin. Par souci de simplification, et pour éviter d'alourdir les notations, on supposera dans ces algorithmes qu'une seule trajectoire est utilisée.

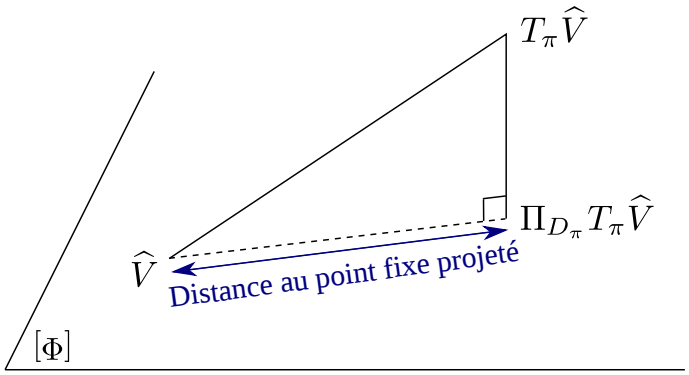


FIGURE 3.1 – Représentation schématique de la projection de $T_\pi \widehat{V}$ sur l'espace d'approximation. L'espace à trois dimensions représente l'espace des fonctions de valeur et le plan représente le sous-espace des fonctions de valeur approchées, qui est défini par les fonctions de base. $T_\pi \widehat{V}$ n'étant pas dans l'espace d'approximation en général, l'équation de Bellman $\widehat{V} = T_\pi \widehat{V}$ n'a pas nécessairement de solution. On effectue donc une projection Π_{D_π} vers cet espace. TD(0) cherche à résoudre l'équation $\widehat{V} = \Pi_{D_\pi} T_\pi \widehat{V}$, c'est-à-dire à déterminer le point fixe de l'opérateur de Bellman T_π projeté.

On considère une trajectoire d'états, actions et récompenses $(s_0, a_0, r_1, s_1, a_1, r_2, s_2, \ldots, s_T)$ générée par la politique π. Nous notons $\widehat{V}_t = \Phi w_t$ la fonction de valeur courante au temps t, c'est-à-dire lors de la visite de l'état s_t. À chaque état s_t visité, la fonction de valeur $\widehat{V}_{t+1}$ est calculée comme une application approximative de $\Pi_{D_\pi} T_\pi$ à la fonction de valeur courante $\widehat{V}_t$. On espère ainsi converger vers une bonne approximation du point fixe de $\Pi_{D_\pi} T_\pi$.

Détaillons la démarche de TD(0). À chaque état s_t observé, on souhaiterait corriger la fonction de valeur dans la direction qui réduit le plus l'erreur quadratique $(V^\pi(s_t) - \phi(s_t)^\mathsf{T} w_t)^2$, mesurée à l'état s_t. Cela revient à effectuer la descende de gradient [5] suivante (Sutton et Barto, 1998) :

$$\begin{aligned} w_{t+1} &= w_t - \frac{1}{2}\alpha_t \nabla_{w_t} \Big[V^\pi(s_t) - \phi(s_t)^\mathsf{T} w_t \Big]^2 \\ &= w_t + \alpha_t \Big[V^\pi(s_t) - \phi(s_t)^\mathsf{T} w_t \Big] \phi(s_t). \end{aligned} \tag{3.1}$$

α_t est un facteur d'apprentissage compris entre 0 et 1 et qui décroît avec le temps. Si α_t est proche de 1, la fonction de valeur sera fortement mise à jour selon les différences temporelles observées. Si α_t est proche de 0, les observations auront moins d'impact sur les mises à jour.

Comme la valeur exacte $V^\pi(s_t)$ est inconnue dans l'équation (3.1), l'idée de TD(0) est de la remplacer par $r_{t+1} + \gamma \phi(s_{t+1})^\mathsf{T} w_t$. Cette quantité est composée d'une part de r_{t+1}, qui une nouvelle observation, et d'autre part de $\phi(s_{t+1})^\mathsf{T} w_t$, qui est la fonction de valeur courante appliquée à l'état suivant, utilisée ici comme une approximation [6] de $V^\pi(s_{t+1})$. Cela revient à appliquer T_π à la fonction de valeur courante dans l'état s_t.

La quantité définie au temps t par

$$\delta(t) = \Big(r_{t+1} + \gamma \phi(s_{t+1})^\mathsf{T} w_t \Big) - \phi(s_t)^\mathsf{T} w_t$$

est appelée la **différence temporelle** à l'état s_t. Elle représente une correction à effectuer dans cet état. Lorsque l'équation de Bellman est vérifiée, elle est en moyenne égale à zéro. TD(0) cherche à annuler les différences temporelles des états observés. La mise à jour que réalise TD(0) à l'état s_t peut ainsi s'écrire :

$$w_{t+1} = w_t + \alpha_t \delta(t) \phi(s_t).$$

5. $\nabla_w f(w)$ désigne le gradient de f par rapport à w, c'est-à-dire le vecteur des dérivées partielles par rapport à chacune des composantes de w : $\nabla_w f(w) = (\frac{\partial f(w)}{\partial w(1)}, \ldots, \frac{\partial f(w)}{\partial w(p)})^\mathsf{T}$.

6. L'idée d'estimer V^π en se basant (en partie) sur la fonction de valeur courante elle-même est appelée *bootstrapping* (Sutton et Barto, 1998). Remarquons qu'il s'agit d'une seconde source d'approximation, la première étant l'utilisation d'une architecture paramétrée pour représenter la fonction de valeur.

Une implémentation de TD(0) est donnée dans l'algorithme 9.

Algorithme 9 TD(0) avec approximation linéaire (Sutton et Barto, 1998)

π : politique à évaluer
$(s_0, a_0, r_1, s_1, a_1, r_2, s_2, \ldots, s_T)$: trajectoire générée avec π
$(\alpha_t)_{t\geq 0}$: facteur d'apprentissage décroissant avec t
$w_0 \leftarrow$ initialisation arbitraire
$t \leftarrow 0$
tant que s_t non terminal **faire**
  $\delta \leftarrow r_{t+1} + \gamma\phi(s_{t+1})^\mathsf{T} w_t - \phi(s_t)^\mathsf{T} w_t$
  $w_{t+1} \leftarrow w_t + \alpha_t \delta \phi(s_t)$
  $t \leftarrow t+1$
fin tant que
retourner w_t

3.3.2 TD(λ)

Avec TD(0), on cherche à résoudre l'équation $\widehat{V} = \Pi_{D_\pi} T_\pi \widehat{V}$ en effectuant à chaque pas de temps de la trajectoire une application approximative (car basée sur des échantillons) de $\Pi_{D_\pi} T_\pi$. Pour cela, dans l'équation (3.1), le terme $V^\pi(s_t)$ est approché par $r_{t+1} + \gamma \widehat{V}_t(s_{t+1})$. Cependant, on peut choisir de l'approcher en considérant aussi la récompense de l'état suivant, c'est-à-dire avec $r_{t+1} + \gamma r_{t+2} + \gamma^2 \widehat{V}_t(s_{t+2})$. L'estimation accorde alors un peu plus de crédit aux récompenses observées et un peu moins à la fonction de valeur courante $\widehat{V}_t$. Au lieu de chercher à calculer à chaque pas de temps $\Pi_{D_\pi} T_\pi \widehat{V}_t$, on cherche à calculer $\Pi_{D_\pi} T_\pi^2 \widehat{V}_t$.

De manière plus générale, il est possible d'employer une estimation de $V^\pi(s_t)$ qui prend en compte les récompenses observées jusqu'à un horizon i donné. Une telle méthode approcherait $V^\pi(s_t)$ par

$$r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^i r_{t+i+1} + \gamma^{i+1} \widehat{V}_t(s_{t+i+1}),$$

ce qui reviendrait à appliquer de façon approximative $\Pi_{D_\pi} T_\pi^{i+1}$ à $\widehat{V}_t$.

Le principe de TD(λ) (Sutton et Barto, 1998) est de faire une moyenne de toutes ces estimations, pondérée de façon géométrique avec un paramètre $\lambda \in [0,1]$ qui contrôle la profondeur de la mise à jour. Ainsi, $V^\pi(s_t)$ est approché par

$$(1-\lambda) \sum_{i=0}^{\infty} \lambda^i \left(r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^i r_{t+i+1} + \gamma^{i+1} \widehat{V}_t(s_{t+i+1}) \right).$$

À chaque pas de temps t, TD(λ) détermine donc $\widehat{V}_{t+1}$ en effectuant de façon approximative le calcul suivant :

$$\Pi_{D_\pi} \left[(1-\lambda) \sum_{i=0}^{\infty} \lambda^i T_\pi^{i+1} \right] \widehat{V}_t.$$

On retrouve dans ce terme exactement l'opérateur T_λ que nous avions introduit dans le contexte de λPI (voir l'équation (1.6) page 27). Nous rappelons ici sa définition : pour une politique π en cours d'évaluation, T_λ est défini pour tout vecteur V par

$$T_\lambda V = (1-\lambda) \left(\sum_{i=1}^{\infty} \lambda^{i-1} T_\pi^i V \right).$$

Autrement dit, TD(λ) applique de façon approximative, à chaque pas de temps t, l'opérateur $\Pi_{D_\pi} T_\lambda$ à la fonction de valeur courante $\widehat{V}_t$. Une application de T_λ réalise un pas de taille réglable par λ dans la direction de V^π. Des applications successives de T_λ permettent de converger vers la valeur de la

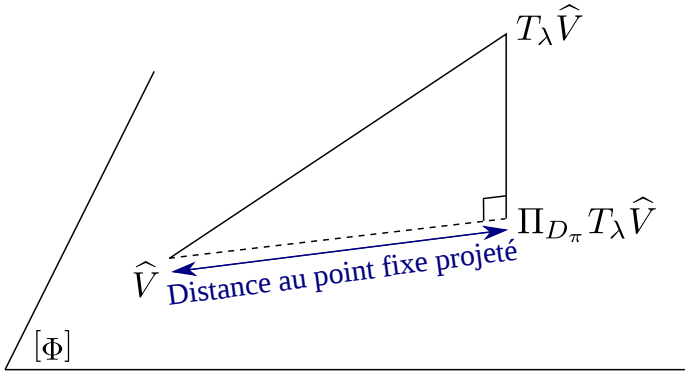


FIGURE 3.2 – Représentation schématique de la projection de $T_\lambda \widehat{V}$ sur l'espace d'approximation. $T_\lambda \widehat{V}$ n'étant pas dans l'espace d'approximation en général, on effectue une projection Π_{D_π} vers cet espace. TD(λ) cherche à résoudre $\widehat{V} = \Pi_{D_\pi} T_\lambda \widehat{V}$, c'est-à-dire à déterminer le point fixe de l'opérateur T_λ projeté. LSTD(λ) et LSPE(λ), qui seront présentés à la section 3.4, cherchent également à obtenir ce point fixe.

politique π, qui est le point fixe de T_λ. En effet, pour la distribution D_π, l'opérateur $\Pi_{D_\pi} T_\lambda$ est une contraction (Sutton et Barto, 1998). L'algorithme cherche donc à résoudre l'équation $\widehat{V} = \Pi_{D_\pi} T_\lambda \widehat{V}$. La figure 3.2 illustre la projection de $T_\lambda \widehat{V}$ sur l'espace d'approximation et la distance qui est minimisée.

Au temps t, la fonction de valeur est mise à jour d'une façon similaire à TD(0) :

$$w_{t+1} = w_t + \alpha_t \delta_\lambda(t) \phi(s_t),$$

où le terme $\delta_\lambda(t)$ remplace la différence temporelle $\delta(t)$ de TD(0) :

$$\delta_\lambda(t) = (1-\lambda) \sum_{i=0}^{\infty} \lambda^i \left(r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^i r_{t+i+1} + \gamma^{i+1} \phi(s_{t+i+1})^\mathsf{T} w_t - \phi(s_t)^\mathsf{T} w_t \right).$$

Comme dans TD(0), il est nécessaire d'employer un facteur d'apprentissage $\alpha_t \in [0,1]$ qui doit décroître avec le temps.

Le paramètre $\lambda \in [0,1]$ contrôle la profondeur des mises à jour de la fonction de valeur. Lorsque $\lambda = 1$, la fonction de valeur est mise à jour avec des différences temporelles qui tiennent compte de toutes les récompenses observées après l'état courant, sans utiliser la fonction de valeur courante comme estimation. Il s'agit alors d'une méthode de Monte-Carlo (Sutton et Barto, 1998). Avec les grandes valeurs de λ, les différences temporelles dépendent fortement des trajectoires observées et ont donc une **variance** importante. On risque alors de commettre une plus grande erreur. A l'inverse, lorsque λ est petit, on accorde plus de crédit à l'estimation courante de la fonction de valeur et moins aux échantillons observés, ce qui réduit la variance mais introduit un **biais** qui ralentit la convergence. Si $\lambda = 0$, seule la récompense immédiate est prise en compte : on a $T_0 = T_\pi$ et il s'agit de TD(0). Ce **compromis biais-variance** de TD(λ) a été étudié analytiquement par Kearns et Singh (2000). Il a été montré (Tsitsiklis et van Roy, 1996) que TD(λ) avec approximation linéaire converge vers une bonne approximation de la fonction de valeur sous réserve que le facteur d'apprentissage α_t soit décroissant et correctement réglé. Nous décrivons TD(λ) dans l'algorithme 10.

Implémentation de TD(λ)

Tel que décrit dans l'algorithme 10, TD(λ) a besoin pour chaque état de calculer des différences temporelles dans le futur (voir la définition de $\delta_\lambda(t)$ à l'équation (3.3.2)). Au niveau de l'implémentation, cela requiert de mémoriser l'ensemble de la trajectoire explicitement et d'attendre la fin d'une trajectoire complète pour pouvoir commencer à mettre à jour la fonction de valeur. Il est possible pour éviter cela d'utiliser la notion des traces d'éligibilité (Sutton et Barto, 1998). Les traces d'éligibilité permettent de propager les récompenses vers les états précédemment visités au fur et à mesure du parcours de

Algorithme 10 TD(λ) avec approximation linéaire, vue en avant théorique (Sutton et Barto, 1998)

π : politique à évaluer
$(s_0, a_0, r_1, s_1, a_1, r_2, s_2, \ldots, s_T)$: trajectoire générée avec π
$(\alpha_t)_{t\geq 0}$: facteur d'apprentissage décroissant avec t
$w_0 \leftarrow$ initialisation arbitraire
$t \leftarrow 0$
tant que s_t non terminal **faire**
  $\delta \leftarrow (1-\lambda)\sum_{i=0}^{\infty} \lambda^i \Big(r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^i r_{t+i+1} + \gamma^{i+1} \phi(s_{t+i+1})^\mathsf{T} w_t - \phi(s_t)^\mathsf{T} w_t \Big)$
  $w_{t+1} \leftarrow w_t + \alpha_t \delta \phi(s_t)$
  $t \leftarrow t+1$
fin tant que
retourner w_t

la trajectoire, et sans avoir à mémoriser celle-ci explicitement. Sutton et Barto (1998) ont montré que l'utilisation de traces d'éligibilité dans TD(λ) était une bonne approximation du calcul théorique présenté plus haut (algorithme 10). Une telle implémentation est donnée dans l'algorithme 11.

Algorithme 11 TD(λ) avec approximation linéaire, vue en arrière (Sutton et Barto, 1998)

π : politique à évaluer
$(s_0, a_0, r_1, s_1, a_1, r_2, s_2, \ldots, s_T)$: trajectoire générée avec π
$(\alpha_t)_{t\geq 0}$: facteur d'apprentissage décroissant avec t
$w_0 \leftarrow$ initialisation arbitraire
$\delta \leftarrow 0$
$t \leftarrow 0$
$z_t \leftarrow \phi(s_t)$
tant que s_t non terminal **faire**
  $\delta \leftarrow \delta + z_t (r_{t+1} + (\gamma \phi(s_{t+1}) - \phi(s_t))^\mathsf{T} w_t)$
  $z_t \leftarrow \lambda \gamma z_t + \phi(s_{t+1})$
  $w_{t+1} \leftarrow w_t + \alpha \delta$
  $t \leftarrow t+1$
fin tant que
retourner w_t

3.3.3 Limites des approches du premier ordre

TD(0) et TD(λ) sont des algorithmes dits du premier ordre, c'est-à-dire que la fonction de valeur approximative est mise à jour seulement via des petites corrections en direction de la fonction de valeur réelle. La complexité par itération a l'avantage d'être linéaire ($O(p)$, où p est la dimension de l'architecture). Mais l'effet de chaque échantillon est assez limité et l'apprentissage nécessite souvent de traiter un grand nombre d'échantillons avant d'être performant. Les données pourraient être exploitées de manière plus efficace. Ensuite, le facteur d'apprentissage α_t est un paramètre qui s'avère souvent difficile à régler en pratique. Pour pallier ces inconvénients, les algorithmes dits du second ordre ou encore avec échantillonnage efficace ont été introduits.

3.4 Approximation linéaire du second ordre

Les approches du second ordre telles que LSTD (Bradtke et Barto, 1996), LSTD(λ) (Boyan, 2002), LSPE(λ) (Nedić et Bertsekas, 2003) cherchent, comme TD(λ), à résoudre l'équation $\widehat{V} = \Pi_{D_\pi} T_\lambda \widehat{V}$, mais

elles ont pour principe, en exploitant efficacement les informations des échantillons, de construire des quantités qui caractérisent cette équation et non de construire la solution directement.

Ces algorithmes exploitent plus efficacement les échantillons, ce qui permet une meilleure vitesse de convergence, au prix d'une complexité supérieure de chaque itération à cause d'un système linéaire à résoudre. Ainsi, chaque itération a typiquement une complexité en $O(p^2)$ au lieu de $O(p)$. Cette complexité s'avère peu gênante en pratique étant donné qu'en général, la dimension de l'architecture linéaire p est très faible comparée au nombre d'états. De plus, les algorithmes du second ordre ne nécessitent pas de facteur d'apprentissage difficile à régler. Il a été argumenté analytiquement et empiriquement (voir par exemple Boyan (2002); Schoknecht (2002); Yu et Bertsekas (2009)) que ces méthodes sont plus stables et peuvent donner de bien meilleures performances que TD(λ). Nous décrivons ici quelques approches significatives de la littérature sur lesquelles nous allons nous appuyer par la suite.

3.4.1 LSTD et LSTD(λ)

Une alternative naturelle du second ordre à TD(λ) est l'algorithme LSTD(λ) (Least-Squares Temporal Differences) proposé par Boyan (2002). Le cas de $\lambda = 0$, intitulé LSTD ou LSTD(0), a été introduit précédemment par Bradtke et Barto (1996).

Boyan (2002) a montré que l'équation $\Phi w = \Pi_{D_\pi} T_\lambda \Phi w$ que l'on cherche à résoudre dans TD(λ) est équivalente à un sytème linéaire noté $Aw = b$, de taille $p \times p$ et d'inconnue w, avec

$$\begin{aligned} A &= E\left[\sum_{t\geq 0} z_t(\phi(s_t) - \phi(t_{t+1}))^\mathsf{T} \middle| a_t = \pi(s_t)\right], \\ b &= E\left[\sum_{t\geq 0} z_t r_t \middle| a_t = \pi(s_t)\right], \\ \text{et} \quad z_t &= E\left[\sum_{i=0}^{t} \lambda^{t-i}\phi(s_t) \middle| a_t = \pi(s_t)\right]. \end{aligned}$$

Il est possible de construire explicitement une estimation $(\widetilde{A}, \widetilde{b})$ de ce système au fur et à mesure du parcours des trajectoires, puis de résoudre ce système pour déterminer la fonction de valeur approximative. Ainsi, au lieu de mettre à jour la fonction de valeur pour chaque état s_t visité, LSTD(λ) construit incrémentalement un système linéaire qui caractérise la solution vers laquelle TD(λ) convergerait.

Nous décrivons cela dans les algorithmes 12 et 13. L'algorithme 12 traite du cas particulier de LSTD (où $\lambda = 0$), et l'algorithme 13 décrit le cas général LSTD(λ). Notons que LSTD(λ) ne nécessite pas de facteur d'apprentissage et n'utilise pas d'estimée initiale de la fonction de valeur.

Algorithme 12 LSTD (Bradtke et Barto, 1996)

π : politique à évaluer
$(s_0, a_0, r_1, s_1, a_1, r_2, s_2, \ldots, s_T)$: trajectoire générée avec π
$\widetilde{A} \leftarrow 0, \widetilde{b} \leftarrow 0$
$t \leftarrow 0$
tant que s_t non terminal **faire**
  $\widetilde{A} \leftarrow \widetilde{A} + \phi(s_t)(\phi(s_t) - \gamma\phi(s_{t+1}))^\mathsf{T}$
  $\widetilde{b} \leftarrow \widetilde{b} + \phi(s_t) r_{t+1}$
  $t \leftarrow t + 1$
fin tant que
$w \leftarrow \widetilde{A}^{-1}\widetilde{b}$
retourner w

La convergence de LSTD a été établie par Bradtke et Barto (1996), et celle de LSTD(λ) par Nedić et Bertsekas (2003).

Notons que si le nombre de fonctions de base p est élevé, le calcul de la matrice $\widetilde{A}$ peut être coûteux, dans la mesure où il requiert une complexité en $O(p^3)$. Il peut être souhaitable de réduire cette complexité.

Algorithme 13 LSTD(λ) (Boyan, 2002)

π : politique à évaluer
$(s_0, a_0, r_1, s_1, a_1, r_2, s_2, \ldots, s_T)$: trajectoire générée avec π
$\widetilde{A} \leftarrow 0, \widetilde{b} \leftarrow 0$
$t \leftarrow 0$
$z_t \leftarrow \phi(s_t)$
tant que s_t non terminal **faire**
  $\widetilde{A} \leftarrow \widetilde{A} + z_t(\phi(s_t) - \gamma\phi(s_{t+1}))^\mathsf{T}$
  $\widetilde{b} \leftarrow \widetilde{b} + z_t r_{t+1}$
  $z_t \leftarrow \lambda\gamma z_t + \phi(s_{t+1})$
  $t \leftarrow t + 1$
fin tant que
$w \leftarrow \widetilde{A}^{-1}\widetilde{b}$
retourner w

Pour cela, au lieu de mettre à jour à chaque itération une estimation de A, on peut maintenir directement une estimation de A^{-1}, grâce à la formule de Sherman-Morrison (voir Golub et Loan, 1996). Pour toute matrice inversible H et tous vecteurs u et v de dimensions compatibles, cette formule établit que

$$(H + uv^\mathsf{T})^{-1} = H^{-1} - \frac{H^{-1}uv^\mathsf{T}H^{-1}}{1 + v^\mathsf{T}H^{-1}u}. \tag{3.2}$$

Ainsi, dans l'algorithme 13 (et de manière similaire dans l'algorithme 12 avec $\lambda = 0$), la mise à jour de $\widetilde{A}$ peut être remplacée par

$$\widetilde{A}^{-1} \leftarrow \widetilde{A}^{-1} - \frac{\widetilde{A}^{-1}z_t(\phi(s_t) - \gamma\phi(s_{t+1}))^\mathsf{T}\widetilde{A}^{-1}}{1 + (\phi(s_t) - \gamma\phi(s_{t+1}))^\mathsf{T}\widetilde{A}^{-1}z_t}$$

et la complexité du calcul de $\widetilde{A}^{-1}\widetilde{b}$ devient $O(p^2)$ au lieu de $O(p^3)$. Afin d'assurer l'inversibilité de la matrice $\widetilde{A}^{-1}$ lors de l'initialisation, on peut fixer la valeur initiale de $\widetilde{A}$ à cI (où c est une constante positive) et donc celle de $\widetilde{A}^{-1}$ à $(1/c)I$.

3.4.2 LSPE(λ)

Nous venons de voir que TD(λ) et LSTD(λ) sont deux algorithmes d'évaluation de la politique qui cherchent à résoudre l'équation $\widehat{V} = \Pi_{D_\pi} T_\lambda \widehat{V}$. Un troisième algorithme plus récent, intitulé LSPE(λ), partage ce même objectif. Il a été proposé par Nedić et Bertsekas (2003) et analysé par Bertsekas *et al.* (2003) puis par Yu et Bertsekas (2009).

Alors que LSTD(λ) construit explictement un système qui caractérise le point fixe projeté de T_λ, LSPE(λ) recherche ce point fixe en réalisant des applications approximatives successives de l'opérateur T_λ à chaque pas de temps d'une trajectoire, comme le fait TD(λ). La différence entre LSPE(λ) et TD(λ) est qu'avec LSPE(λ), l'application de T_λ est approchée d'une manière qui exploite plus efficacement les informations des échantillons. Nous décrivons ici de quelle manière LSPE(λ) procède.

Rappelons que $T_\lambda \widehat{V}_t$ peut s'écrire sous la forme d'une espérance (voir l'équation 1.8 page 27) :

$$T_\lambda \widehat{V}_t = \widehat{V}_t + \Delta_t$$

avec

$$\forall s \in \mathcal{S} \quad \Delta_t(s) = E\left[\sum_{t=0}^{\infty}(\lambda\gamma)^t \delta(t) \,\middle|\, s_0 = s, a_t = \pi(s_t)\right]$$

où les $\delta(t)$ sont les différences temporelles données par $\delta(t) = r_{t+1} + \gamma\widehat{V}_t(s_{t+1}) - \widehat{V}_t(s_t)$.

LSPE(λ) cherche la fonction de valeur $\widehat{V}_{t+1}$ qui minimise $\widehat{V}_{t+1} - (\widehat{V}_t + \Delta_t)$ à chaque état s_t visité. Δ_t est estimé à partir de la trajectoire jusqu'au temps t. Plus précisément, on calcule

$$\widehat{V}_{t+1} = \arg\min_{\widehat{V}} \sum_{m=0}^{t} \left(\widehat{V}(s_m) - \widehat{V}_t(s_m) - \sum_{n=m}^{t} (\gamma\lambda)^{n-m} \delta(n) \right)^2 .$$

Pour chaque état observé s_t, LSPE(λ) détermine $\widehat{V}_{t+1}$ en effectuant une résolution standard de ce problème aux moindres carrés.

Cependant, au lieu de recalculer une nouvelle résolution aux moindres carrés à chaque pas de temps de la trajectoire, les auteurs de LSPE(λ) (Nedić et Bertsekas, 2003) ont montré qu'il est possible d'implémenter les résolutions de manière incrémentale en maintenant à jour un sytème linéaire. On peut ainsi, comme dans LSTD(λ), attendre la fin de la trajectoire pour résoudre le système. Il se trouve que le système linéaire que mantient LSPE(λ) a des similitudes avec celui que construit LSTD(λ). Les deux algorithmes sont en fait assez proches : Yu et Bertsekas (2009) ont montré que LSPE(λ) et LSTD(λ) ont la même vitesse de convergence, et qu'ils convergent l'un vers l'autre à une vitesse plus rapide que celle avec laquelle ils atteignent leur limite commune. Nedić et Bertsekas (2003) avaient auparavant établi la convergence de LSPE(λ) selon certaines conditions que nous ne détaillons pas ici. LSPE(λ) est décrit dans l'algorithme 14.

Algorithme 14 LSPE(λ) (Nedić et Bertsekas, 2003)

π : politique à évaluer
$(s_0, a_0, r_1, s_1, a_1, r_2, s_2, \ldots, s_T)$: trajectoire générée avec π
$w_0 \leftarrow$ initialisation arbitraire
$\widetilde{A} \leftarrow 0, \widetilde{b} \leftarrow 0$
$t \leftarrow 0$
$z_t \leftarrow \phi(s_t)$
tant que s_t non terminal **faire**
  $\widetilde{A} \leftarrow \widetilde{A} + \phi(s_t)\phi(s_t)^\mathsf{T}$
  $\widetilde{b} \leftarrow \widetilde{b} + z_t((\gamma\phi(s_{t+1})^\mathsf{T} - \phi(s_t)^\mathsf{T})w_t + r_{t+1})$
  $z_{t+1} \leftarrow \gamma\lambda z_t + \phi(s_{t+1})$
  $w_{t+1} \leftarrow w_t + \alpha(\widetilde{A}^{-1}\widetilde{b} - w_t)$
  $t \leftarrow t + 1$
fin tant que
retourner w_t

De manière analogue à LSTD(λ), on peut là aussi maintenir directement une estimation de A^{-1} en utilisant la formule de Sherman-Morrison. La mise à jour de $\widetilde{A}$ est alors remplacée dans l'algorithme 14 par

$$\widetilde{A}^{-1} \leftarrow \widetilde{A}^{-1} - \frac{\widetilde{A}^{-1}\phi(s_t)\phi(s_t)^\mathsf{T}A^{-1}}{1 + \phi(s_t)^\mathsf{T}\widetilde{A}^{-1}\phi(s_t)}$$

et la complexité du calcul de la nouvelle fonction de valeur est réduite à $O(p^2)$.

Lien avec TD(λ)

Revenons sur la minimisation aux moindres carrés que LSPE(λ) résout à l'état s_t pour calculer $\Pi_{D_\pi} T_\lambda \widehat{V}_t$ à partir des échantillons observés jusqu'au temps t :

$$\widehat{V}_{t+1} = \arg\min_{\widehat{V}} \sum_{m=0}^{t} \left(\widehat{V}(s_m) - \widehat{V}_t(s_m) - \sum_{n=m}^{t} (\gamma\lambda)^{n-m} \delta(n) \right)^2 .$$

Bertsekas *et al.* (2003) ont montré que le calcul que réalise TD(λ) au temps t revient en fait en moyenne à mettre à jour $\widehat{V}_{t+1}$ en direction du gradient de la somme ci-dessus. Ainsi, TD(λ) peut être vu comme

une approximation de LSPE(λ) dans laquelle on effectue une petite correction en direction du minimum de cette somme au lieu de la minimiser entièrement.

3.4.3 Approximate λPI

Les algorithmes présentés précédemment dans ce chapitre se focalisent sur le problème d'évaluer une politique fixée π. Même si TD(λ), LSTD, LSTD(λ) et LSPE(λ) peuvent être utilisés comme phase d'évaluation dans un schéma d'itération sur les politiques (sous réserve de générer de nouveaux échantillons à chaque itération), nous avons vu dans les chapitres 1 et 2 que lorsque l'on cherche à construire une politique itérativement, il n'est pas nécessaire d'évaluer complètement chaque politique intermédiaire : l'algorithme peut changer de politique plus tôt (voir la figure 1.2 page 28) et il est alors dit **optimiste**.

On peut, dans le cas approché, envisager un schéma d'itération sur les politiques optimiste, c'est-à-dire où la politique courante est évaluée de façon incomplète avant de passer à la politique suivante. Pour cela, un algorithme naturel consiste à appliquer une seule fois l'opérateur $\Pi_{D_\pi} T_\lambda$ à la fonction de valeur courante $\widehat{V}_k$. On peut ainsi considérer une variante de LSPE(λ) qui appliquerait $\Pi_{D_\pi} T_\lambda$ une seule fois (et de façon approximative), après avoir parcouru l'intégralité d'une trajectoire, au lieu de l'appliquer répétitivement à chaque état visité. L'objectif est ici de rechercher le vecteur $\widehat{V}_{k+1}$ qui correspond à une seule application de $\Pi_{D_\pi} T_\lambda$, c'est-à-dire qui vérifie $\widehat{V}_{k+1} = \Pi_{D_\pi} T_\lambda \widehat{V}_k$.

Une telle variante a en fait été proposée avant LSPE(λ) par les auteurs de λPI (Bertsekas et Ioffe, 1996) sous le nom de Approximate λ-Policy Iteration (AλPI). Après avoir parcouru la trajectoire entière, AλPI calcule la valeur $\widehat{V}_{k+1}$ qui minimise au sens des moindres carrés la même quantité que dans LSPE(λ) :

$$\widehat{V}_{k+1} = \arg\min_{\widehat{V}} \sum_{m=0}^{T} \left(\widehat{V}(s_m) - \widehat{V}_k(s_m) - \sum_{n=m}^{T} (\gamma\lambda)^{n-m} \delta(n) \right)^2 .$$

Ensuite, on calcule la politique gourmande par rapport à $\widehat{V}_{k+1}$, on génère de nouveaux échantillons et on répète le processus.

Les auteurs soulignent que dans le cas approché, lorsque la fonction de valeur est approximative et qu'elle est estimée à l'aide d'échantillons, prendre $\lambda < 1$ (c'est-à-dire effectuer une évaluation incomplète ou encore optimiste) peut s'avérer bénéfique. Nous avions vu que, dans la version exacte de λPI, la vitesse de convergence asymptotique se dégrade lorsque λ diminue (Bertsekas et Ioffe, 1996). Cela dit, la convergence asymptotique correspond au moment où la politique obtenue est optimale et où il ne reste plus qu'à affiner la fonction de valeur ; ici, le fait que la vitesse de convergence asymptotique se dégrade lorsque $\lambda < 1$ va s'avérer peu crucial dans la mesure où de toute manière, on ne peut en général pas atteindre une politique optimale. Ensuite, l'estimation qui est calculée par AλPI ou LSPE(λ) à partir des échantillons a une certaine variance qui va pouvoir être diminuée grâce à l'optimisme. En effet, considérons à nouveau l'écriture de $T_\lambda \widehat{V}_k$ en fonction des différences temporelles : $T_\lambda \widehat{V}_k = \widehat{V}_k + \Delta_k$, avec

$$\forall s \in \mathcal{S} \quad \Delta_k(s) = E\left[\sum_{t=0}^{\infty} (\lambda\gamma)^t \delta(t) \middle| s_0 = s, a_t = \pi_{k+1}(s_t) \right].$$

Cette expression sous forme d'espérance met en évidence le fait que le paramètre λ, tout en réglant l'optimisme de l'algorithme, possède une influence de type compromis biais-variance. Lorsque $\lambda = 1$, on peut voir que $\Delta_k = V^{\pi_{k+1}} - \widehat{V}_k$: ainsi, on cible la vraie valeur de π_{k+1} et il n'y a pas de biais. Cependant, comme Δ_k est estimé à partir d'une base d'échantillons, on peut voir sur l'équation ci-dessus que l'horizon de la somme à estimer est plus important pour les grandes valeurs de λ. La variance de l'estimation risque alors de pénaliser l'algorithme. En revanche, lorsque $\lambda < 1$, cette variance est réduite, mais on introduit un biais lié au fait que le vecteur $T_\lambda \widehat{V}_k$ ciblé à chaque itération n'est plus la valeur de π_{k+1}. Le nombre d'itérations nécessaires sera donc plus important, mais chaque itération sera moins sensible à la variance de l'estimation.

Une illustration intuitive de ce compromis biais-variance est donnée sur la figure 3.3. Lorsque $\lambda = 1$, à l'itération k, on cible $V^{\pi_{k+1}}$ mais en raison de la variance de l'estimation, on commet une erreur ϵ_{k+1} relativement importante. Lorsque $\lambda < 1$, on effectue une évaluation optimiste (incomplète) : on ne cible

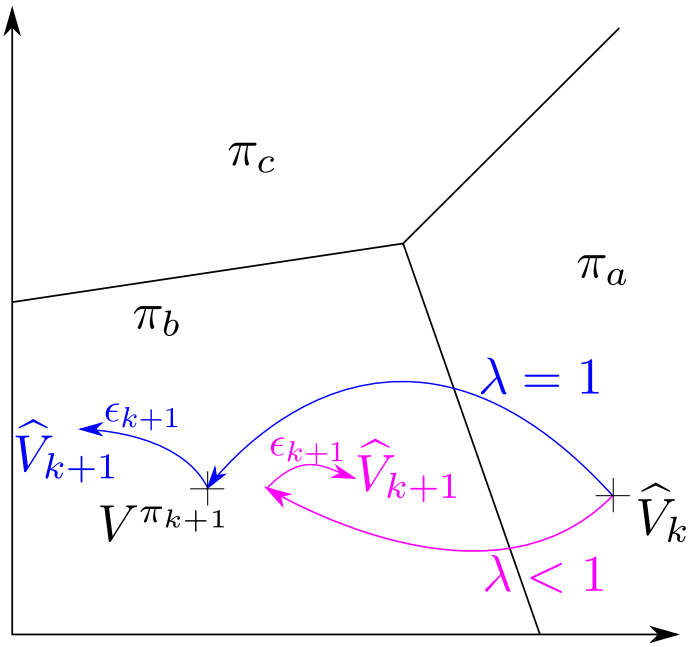


FIGURE 3.3 – Illustration du compromis biais-variance de AλPI dans la partition des politiques gloutonnes. Si $\lambda < 1$, l'algorithme est optimiste : la fonction de valeur ciblée n'est plus $V^{\pi_{k+1}}$ mais l'estimation est plus précise.

plus $V^{\pi_{k+1}}$ mais $T_\lambda \widehat{V}_k$. Cela introduit un biais lié au fait que l'on ne cible plus la vraie fonction de valeur, mais la nouvelle cible est atteinte plus précisément. En fait, grâce à cette réduction de la variance de l'estimation, si le biais n'est pas très important, il se peut même que la fonction de valeur approximative $\widehat{V}_{k+1}$ obtenue soit finalement plus proche de la vraie fonction de valeur $V^{\pi_{k+1}}$ que ce que l'on aurait obtenu avec $\lambda = 1$.

Une contrainte de AλPI est que, comme pour les algorithmes présentés précédemment, l'évaluation est on-policy : à chaque changement de politique, on doit regénérer de nouveaux échantillons. Dans le chapitre 4, nous proposons une autre manière d'approcher λPI, cette fois de façon off-policy. Nous généralisons pour cela les idées de LSPI (Lagoudakis et Parr, 2003), l'algorithme que nous présentons maintenant.

3.4.4 LSPI

Les algorithmes présentés plus haut (TD(λ), LSTD, LSTD(λ), LSPE(λ) et AλPI), lorsqu'ils sont placés dans un schéma d'itération sur les politiques, ont l'inconvénient de nécessiter de générer de nouveaux échantillons à chaque changement de politique. Least-Squares Policy Iteration (LSPI) (Lagoudakis et Parr, 2003) propose une solution à ce problème. Il s'agit d'un algorithme d'itération sur les politiques avec approximation du second ordre qui utilise des fonctions de valeurs définies sur l'espace d'états-actions (fonctions de valeur Q) et où la phase d'évaluation de la politique s'appuie sur LSTD (Bradtke et Barto, 1996) (c'est-à-dire LSTD(λ) avec $\lambda = 0$).

L'utilisation de fonctions de valeur Q combiné avec LSTD permet en effet d'évaluer une politique de façon off-policy, c'est-à-dire à l'aide d'échantillons qui peuvent avoir été générés avec une autre politique, et ces échantillons peuvent se limiter à des transitions individuelles de la forme (s, a, r', s') au lieu d'être des trajectoires complètes.

LSTD (voir la section 3.4.1) cherche à résoudre l'équation $\widehat{V} = \Pi_{D_\pi} T_0 \widehat{V}$ (rappelons que $\lambda = 0$ et que par ailleurs, $T_0 = T_\pi$). Avec l'utilisation de fonctions de valeurs Q, le fait que l'évaluation puisse être off-policy signifie que la projection orthogonale utilisée peut cette fois être pondérée par une distribution quelconque. Dans le cas on-policy, il s'agissait nécessairement de la distribution stationnaire associée à

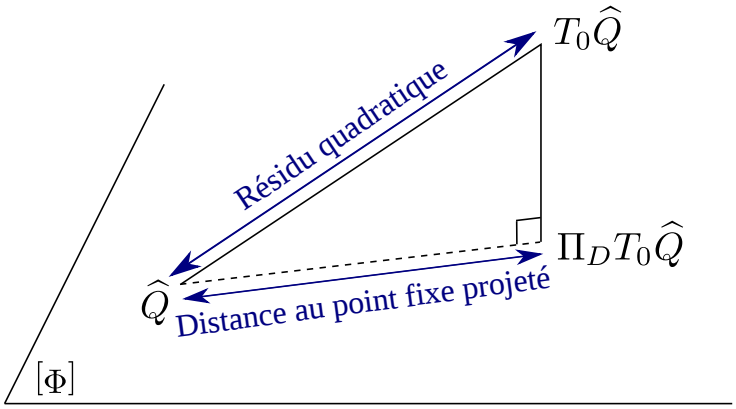


FIGURE 3.4 – Représentation schématique des deux méthodes LSTDQ et LSBRQ. LSTDQ cherche la fonction de valeur approchée qui est le point fixe de $T_0 = T_\pi$ suivi d'une projection Π_D sur l'espace des fonctions de valeur approximatives, alors que LSBRQ cherche la fonction de valeur qui minimise la distance entre elle-même et une application de l'opérateur $T_0 = T_\pi$.

la politique π en cours d'évaluation. Cette méthode d'évaluation, intitulée LSTDQ (Lagoudakis et Parr, 2003), cherche à résoudre l'équation $\widehat{Q} - \Pi_D T_0 \widehat{Q}$. LSTDQ est décrit à l'algorithme 15. Au niveau de l'implémentation, LSTDQ diffère de LSTD (algorithme 12) uniquement par l'utilisation de fonctions de valeur Q.

Algorithme 15 LSTDQ (Lagoudakis et Parr, 2003)

π : politique à évaluer
$\widetilde{A} \leftarrow 0, \widetilde{b} \leftarrow 0$
pour chaque échantillon (s, a, r', s') **faire**
  $\widetilde{A} \leftarrow \widetilde{A} + \phi(s,a)\Big(\phi(s,a) - \gamma\phi(s', \pi(s'))\Big)^\mathsf{T}$
  $\widetilde{b} \leftarrow \widetilde{b} + \phi(s,a)r'$
fin pour
$w \leftarrow \widetilde{A}^{-1}\widetilde{b}$
retourner w

Algorithme 16 LSBRQ (Lagoudakis et Parr, 2003)

π : politique à évaluer
$\widetilde{A} \leftarrow 0, \widetilde{b} \leftarrow 0$
pour chaque échantillon (s, a, r', s', s'') **faire**
  $\widetilde{A} \leftarrow \widetilde{A} + \Big(\phi(s,a) - \gamma\phi(s'', \pi(s''))\Big)\Big(\phi(s,a) - \gamma\phi(s', \pi(s'))\Big)^\mathsf{T}$
  $\widetilde{b} \leftarrow \widetilde{b} + \Big(\phi(s,a) - \gamma\phi(s'', \pi(s''))\Big)r'$
fin pour
$w \leftarrow \widetilde{A}^{-1}\widetilde{b}$
retourner w

Une alternative à LSTDQ pour estimer la fonction de valeur de la politique courante est la méthode LSBRQ (algorithme 16). LSBRQ cherche à minimiser directement la norme quadratique du résidu de Bellman $\|\widehat{Q} - T_0\widehat{Q}\|^2$. La figure 3.4 donne une illustration de ces deux critères à minimiser (voir aussi Lagoudakis et Parr, 2003). LSBRQ a l'inconvénient de nécessiter des échantillons de la forme (s, a, r', s', s'') où s' et s'' sont deux réalisations indépendantes de l'action a depuis l'état s (Sutton et Barto, 1998). Ainsi, LSBRQ est plus contraignant à utiliser en pratique, à moins de disposer d'un modèle

génératif. Nous reviendrons plus en détail sur ces deux méthodes dans le chapitre 4 où nous proposons une généralisation de LSPI.

Comme avec LSTD(λ) et LSPE(λ), il est également possible de maintenir à jour directement une estimation de A^{-1} pour réduire la complexité du calcul de $\widetilde{A}^{-1}\widetilde{b}$, et ce avec les deux méthodes. Les formules correspondantes seront détaillées dans le chapitre 4 dans le contexte plus général de LSλPI.

De plus, si un modèle du PDM est connu, il est possible d'exploiter cette connaissance. Les échantillons se limitent alors à des couples états-actions (s, a) et on remplace dans les algorithmes ci-dessus les termes de la forme $\phi(s', \pi(s'))$ et r' par leur espérance respective $\sum_{s' \in \mathcal{S}} P(s, a, s')\phi(s', \pi(s'))$ et $R(s, a)$.

Au final, LSPI est résumé dans l'algorithme 17, pour le choix d'une méthode d'évaluation donnée : LSTDQ ou LSBRQ.

Algorithme 17 LSPI (Lagoudakis et Parr, 2003)

$\mathcal{D} \leftarrow$ ensemble d'échantillons de la forme $\begin{cases} (s, a, r', s') & \text{avec LSTDQ} \\ (s, a, r', s', s'') & \text{avec LSBRQ} \end{cases}$
$k \leftarrow 0, w_0 \leftarrow$ initialisation arbitraire
répéter
  $\pi_{k+1} \leftarrow \text{glouton}(\Phi w_k)$
  $w_{k+1} \leftarrow \begin{cases} \text{LSTDQ}(\mathcal{D}, \pi_{k+1}) \\ \text{ou} \\ \text{LSBRQ}(\mathcal{D}, \pi_{k+1}) \end{cases}$
  $k \leftarrow k + 1$
jusqu'à $\|w_k - w_{k-1}\|_\infty < \epsilon$

Synthèse et conclusion

Nous avons présenté dans ce chapitre différents algorithmes d'évaluation de la fonction de valeur. La figure 3.5 propose une vue synthétique des différentes approches, organisées selon la manière dont elles calculent la fonction de valeur. Nous détaillons ici cette vue schématique. On considère une équation de Bellman générale $\Phi w = T_\lambda \Phi w$, équivalente à l'équation de Bellman usuelle $\Phi w = T_\pi \Phi w$. Comme elle n'a pas nécessairement de solution dans l'espace d'approximation, on peut soit minimiser le résidu quadratique $\|\Phi w - T_\lambda \Phi w\|^2$, soit rechercher le point fixe de l'opérateur T_λ auquel on applique une certaine projection Π_D. La première possibilité n'est connue que pour $\lambda = 0$ dans la littérature, avec LSBRQ (Lagoudakis et Parr, 2003). La seconde possibilité se divise en deux cas selon la distribution utilisée pour réaliser la projection. Si l'on ne met pas de contrainte sur cette distribution D, par exemple pour utiliser des échantillons off-policy, seul le cas $\lambda = 0$ a été proposé : il s'agit de l'algorithme LSTDQ (Lagoudakis et Parr, 2003). Si l'on considère en revanche la distribution stationnaire D_π associée à la politique à évaluer π, il existe alors plusieurs approches, qui s'appuient sur une trajectoire $(s_0, a_0, r_1, s_1, \ldots, s_T)$ générée avec la politique π. s_T est l'état terminal de la trajectoire. Ces approches cherchent à résoudre l'équation $\Phi w = \Pi_{D_\pi} T_\lambda \Phi w$. LSTD($\lambda$) (Boyan, 2002) construit incrémentalement, au fur et à mesure du parcours de la trajectoire, une estimation de deux quantités A et b qui caractérisent le point fixe de $\Pi_{D_\pi} T_\lambda$, puis calcule finalement $w = \widetilde{A}^{-1}\widetilde{b}$. Une autre idée possible est de rechercher ce point fixe en effectuant des applications successives (et approximatives) de $\Pi_{D_\pi} T_\lambda$, à la manière de Value Iteration. Cette méthode est justifiée par le fait que pour la distribution D_π, l'opérateur $\Pi_{D_\pi} T_\lambda$ est une contraction. Appliquer cet opérateur de façon approximative à partir des échantillons $(s_0, a_0, r_1, \ldots, s_t)$ observés jusqu'au temps t revient à minimiser le critère suivant :

$$\min_w \sum_{m=0}^{t} \left(\Phi w(s_m) - \Phi \bar{w}(s_m) - \sum_{n=m}^{t} (\gamma\lambda)^{n-m} \delta(s_n) \right)^2$$

où $\Phi\bar{w}$ représente la fonction de valeur précédente. TD(λ) (Sutton et Barto, 1998) peut être vu comme un algorithme qui, pour chaque t de 0 à T, calcule le gradient de cette somme et corrige la fonction

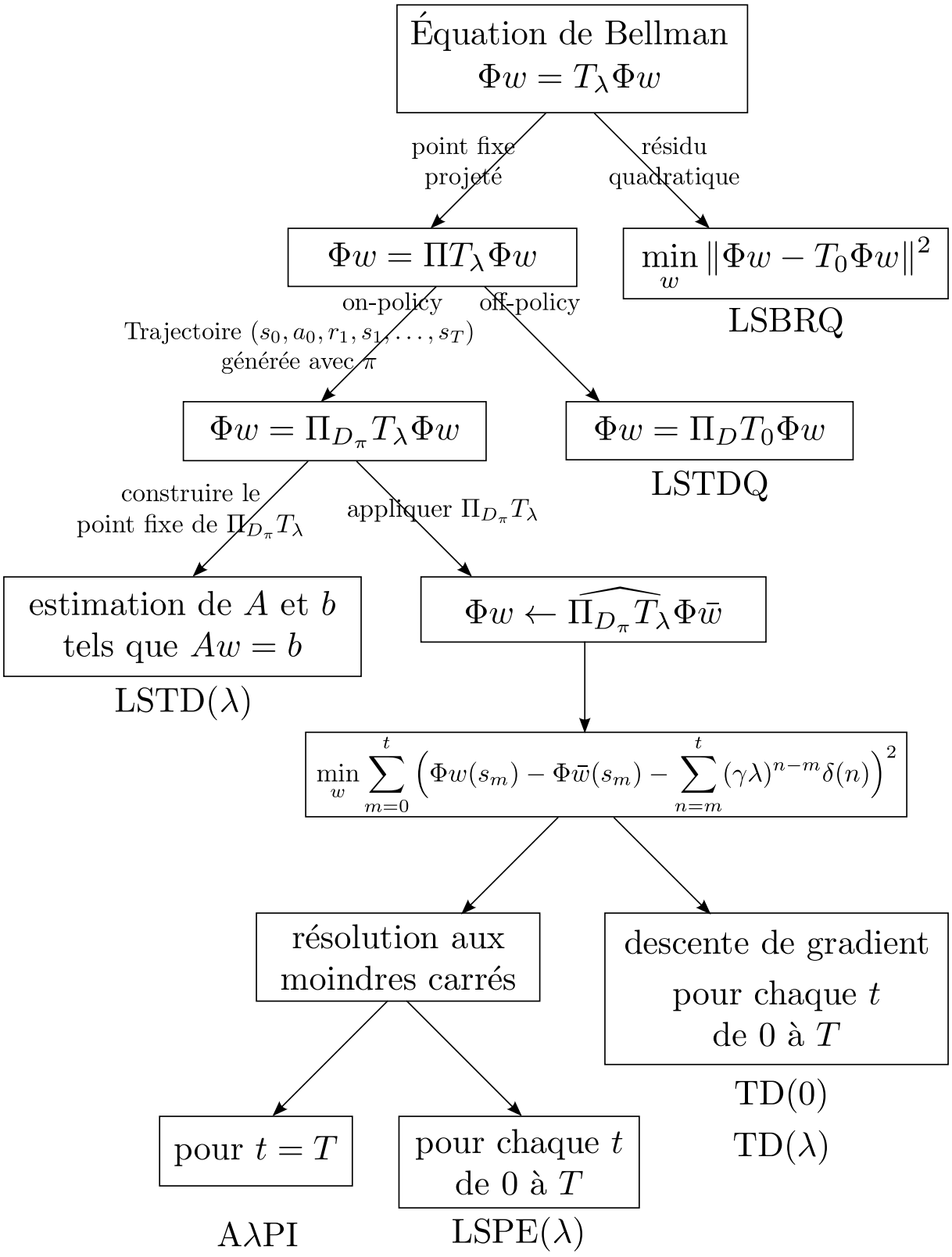


FIGURE 3.5 – Vue d'ensemble des algorithmes d'approximation linéaire mentionnés dans le chapitre 3 (voir le texte pour l'explication détaillée du schéma). On considère une politique π à évaluer et on recherche une fonction de valeur approximative Φw. $\Phi \bar{w}$ est une estimation précédente de la fonction de valeur. D_π désigne la distribution stationnaire associée à la politique π tandis que D peut désigner une distribution quelconque.

Synthèse des méthodes d'évaluation d'une politique π_{k+1}					
Algorithme d'évaluation	Critère considéré	Évaluation optimiste	Compromis biais-variance (λ)	Off-policy	Échantillonnage efficace
TD(0)	$\widehat{V} = \Pi_{D_\pi} T_0 \widehat{V}$				
TD(λ)	$\widehat{V} = \Pi_{D_\pi} T_\lambda \widehat{V}$		×		
LSTD(λ)			×		×
LSPE(λ)			×		×
AλPI (phase d'évaluation)	$\widehat{V} = \Pi_{D_\pi} T_\lambda \widehat{V}_k$	×	×		×
LSTDQ	$\widehat{Q} = \Pi_D T_0 \widehat{Q}$			×	×
LSBRQ	$\min \|\widehat{Q} - T_0 \widehat{Q}\|^2$			×	×

TABLE 3.1 – Synthèse des principales caractéristiques des méthodes d'évaluation de la fonction de valeur. Chaque méthode est ici vue comme un choix possible pour la phase d'évaluation de la politique dans un schéma d'itération sur les politiques à l'itération k. La politique à évaluer, notée π_{k+1}, est la politique gloutonne par rapport à la valeur précédente $\widehat{V}_k$ ou $\widehat{Q}_k$.

de valeur en direction de ce gradient, avec un certain pas d'apprentissage qui peut être délicat à régler. LSPE(λ) (Nedić et Bertsekas, 2003), quant à lui, minimise cette somme par une résolution standard aux moindres carrés (et répète lui aussi l'opération pour chaque t de 0 à T). Le calcul est effectué de façon incrémentale pour éviter de devoir résoudre un système linéaire à chaque itération. TD(λ) et LSPE(λ) réalisent donc tous deux des applications répétées de $\Pi_{D_\pi} T_\lambda$, ces applications étant approchées de deux façons différentes. Enfin, l'algorithme AλPI (Bertsekas et Tsitsiklis, 1996) réalise la même résolution aux moindres carrés que LSPE(λ), mais dans un contexte d'itération sur les politiques optimiste. Il résout le système aux moindres carrés seulement une fois, à la fin de la trajectoire, c'est-à-dire pour $t = T$ uniquement. Il applique donc $\Pi_{D_\pi} T_\lambda$ une seule fois (de façon approchée). Puis il calcule la politique gloutonne par rapport à la fonction de valeur obtenue, génère une nouvelle trajectoire avec cette politique et recommence le processus.

Calculer une politique

Parmi les algorithmes mentionnés dans ce chapitre, certains sont conçus pour s'insérer dans un contexte d'itération sur les politiques et d'autres se concentrent d'abord sur l'évaluation de la politique. Les principales caractéristiques de ces algorithmes sont mentionnées dans le tableau 3.1.

TD(λ), LSTD(λ), LSPE(λ) et AλPI sont obligatoirement on-policy : ils estiment la fonction de valeur à partir d'une trajectoire générée avec la politique à évaluer. De ce fait, il peut être délicat d'utiliser ces algorithmes dans un schéma d'itération sur les politiques. Avec LSPI, on peut se passer de trajectoires complètes générées avec la politique à évaluer car seule la transition immédiate est prise en compte dans LSTDQ et LSBRQ. Les états précédemment visités sont oubliés, contrairement à TD(λ), LSTD(λ), LSPE(λ) et AλPI où un paramètre λ contrôle l'éligibilité de chaque état. L'inconvénient est qu'en l'absence du paramètre λ, on ne bénéficie plus d'un compromis biais-variance réglable. Il serait possible de définir LSTDQ(λ) comme étant LSTD(λ) avec des fonctions de valeurs Q, mais on perdrait le côté off-policy : il faudrait cette fois utiliser des trajectoires complètes générées avec la politique courante. L'approche que nous proposons dans le chapitre suivant ajoute un compromis biais-variance à LSPI via le paramètre λ de λPI, tout en continuant à itérer sur les politiques de manière off-policy.

Chapitre 4

LSλPI : Optimisme et compromis biais-variance pour le contrôle optimal

Dans le chapitre précédent, nous avons présenté quelques algorithmes de la littérature pour évaluer une politique : TD(λ) (Sutton et Barto, 1998), LSTD (Bradtke et Barto, 1996), LSTD(λ) (Boyan, 2002), LSPE(λ) (Yu et Bertsekas, 2009), ou calculer une politique : AλPI (Bertsekas et Ioffe, 1996) et LSPI (Lagoudakis et Parr, 2003). Nous développons dans ce chapitre une nouvelle approche qui cherche à regrouper plusieurs caractéristiques intéressantes de ces algorithmes.

- **Second ordre** : nous souhaitons que les informations des échantillons soient exploitées de façon efficace par l'algorithme, par opposition à TD(λ).
- **Itération sur les politiques** : nous nous intéressons à itérer sur les politiques et non à évaluer une politique fixée.
- **Optimisme** : nous voulons pouvoir changer de politique avant d'avoir entièrement évalué la politique courante, afin de réduire la variance de l'estimation comme le fait AλPI (Bertsekas et Ioffe, 1996).
- **Compromis biais-variance** : un compromis biais-variance réglable par le paramètre λ permet d'améliorer la qualité de l'estimation dans AλPI, TD(λ), LSTD(λ) et LSPE(λ). (Dans le cas de AλPI, le paramètre λ permet également de régler l'optimisme.)
- **Évaluation off-policy** : dans LSPI, la politique peut être évaluée à l'aide d'échantillons individuels d'horizon 1. Des trajectoires complètes ne sont pas nécessaires et cela permet d'évaluer les politiques de manière off-policy. On peut itérer sur les politiques sans avoir à regénérer des échantillons lorsque la politique change. Dès lors qu'elles utilisent un paramètre λ pour faire un compromis biais-variance, les approches de l'état de l'art telles que LSTD(λ), LSPE(λ) et AλPI ont besoin d'estimer la valeur de la politique de façon on-policy, en utilisant des trajectoires générées avec la politique à évaluer.

Après avoir évoqué des caractéristiques qui nous ont paru intéressantes dans les algorithmes de la littérature (itération sur les politiques, optimisme, compromis biais-variance, évaluation du second ordre, off-policy et donc possibilité d'itérer sur les politiques sans regénérer des échantillons), nous pouvons dire de l'algorithme que nous allons présenter, Least-Squares λ Policy Iteration (LSλPI), qu'il est à notre connaissance le premier à toutes les posséder. LSλPI peut être vu comme une généralisation de LSPI avec un paramètre λ qui établit un compromis biais-variance et ajoute de l'optimisme, comme une version off-policy de λPI, ou comme une version avec itération sur les politiques optimiste et off-policy de LSPE(λ).

4.1 L'algorithme LSλPI

Pour mettre au point notre approche, nous nous sommes intéressés en particulier à l'algorithme λPI (Bertsekas et Ioffe, 1996). Rappelons que dans λPI, un paramètre $\lambda \in [0, 1]$ permet d'effectuer une évaluation optimiste de la politique courante, c'est-à-dire de changer de politique avant de l'avoir entièrement évaluée. Lorsque $\lambda < 1$, un biais dû à cette évaluation incomplète dégrade la vitesse de

convergence asymptotique. Dans le cas exact, comme on ne fait pas d'estimation à l'aide d'échantillons, le paramètre λ n'a que peu d'utilité car rien ne vient compenser ce biais : la convergence asymptotique est plus lente et il n'y a pas de contrepartie. En pratique, comme nous l'avons vu au chapitre 2, c'est alors Modified Policy Iteration, plus simple que λPI, qui induit une meilleure vitesse de convergence.

C'est dans le cas approché, lorsque la fonction de valeur est approximative et qu'elle est estimée à l'aide d'échantillons, que diminuer λ va s'avérer intéressant. D'abord, le fait que la vitesse de convergence asymptotique se dégrade lorsque $\lambda < 1$ va s'avérer peu pénalisant dans la mesure où, dans le cas approché, on ne peut en général pas atteindre une politique optimale. La vitesse de convergence asymptotique est donc ici un critère moins crucial. Ensuite, nous avons vu dans le chapitre 3 que l'estimation qui est calculée par λPI à partir des échantillons a une certaine variance qui peut être diminuée grâce à λ.

4.1.1 Idée générale

Dans LSPI (Lagoudakis et Parr, 2003), le vecteur w_{k+1} est calculé à chaque itération de manière à ce que Φw_{k+1} approche la fonction de valeur de π_{k+1}, c'est-à-dire le point fixe de $T_{\pi_{k+1}}$. En d'autres termes, que ce soit via LSTDQ ou LSBRQ, LSPI détermine un w_{k+1} qui vérifie

$$T_{\pi_{k+1}} \Phi w_{k+1} \simeq \Phi w_{k+1}.$$

La démarche que nous proposons ici, intitulée Least-Squares λ Policy Iteration (LSλPI) consiste à généraliser LSPI en y ajoutant le paramètre λ de λPI. Comme dans LSPI, nous considérons uniquement des fonctions de valeurs définies sur l'espace des couples états-actions. Rappelons que l'opérateur M_k (défini à l'équation 1.5 page 26 pour les fonctions de valeur V) correspond à une évaluation amortie de l'opérateur de Bellman $T_{\pi_{k+1}}$. Redéfinissons M_k dans le cas des fonctions de valeurs Q :

$$M_k Q = (1-\lambda) T_{\pi_{k+1}} Q_k + \lambda T_{\pi_{k+1}} Q. \tag{4.1}$$

On peut remarquer dans les algorithmes 4 et 6 (pages 25 et 26) que la seule différence entre Policy Iteration et λPI en version exacte est l'opérateur dont on calcule le point fixe : il s'agit de l'opérateur de Bellman $T_{\pi_{k+1}}$ dans le cas de Policy Iteration, et de l'opérateur M_k dans le cas de λPI. L'idée de LSλPI est donc de rechercher non pas le point fixe de $T_{\pi_{k+1}}$, mais celui de l'opérateur plus général M_k. Il s'agit donc de déterminer un w_{k+1} tel que

$$M_k \Phi w_{k+1} \simeq \Phi w_{k+1}.$$

Ainsi, on ne cherche plus à estimer la valeur $Q^{\pi_{k+1}}$, mais le vecteur Q_{k+1} que λPI calculerait en version exacte. LSPI devient un cas particulier de LSλPI pour lequel $\lambda = 1$. LSPI possède plusieurs caractéristiques intéressantes que LSλPI conserve naturellement : l'échantillonnage efficace (il s'agit d'une méthode du second ordre), l'évaluation off-policy de la fonction de valeur, qui permet de réutiliser les mêmes échantillons malgré les changements de politique, et le fait que le modèle du PDM soit optionnel mais puisse être exploité s'il est disponible. LSλPI ajoute à cela les caractéristiques de λPI discutées plus haut : le compromis biais-variance, qui peut améliorer la qualité de l'estimation, et l'évaluation optimiste de la fonction de valeur.

4.1.2 Méthode de projection du point fixe : LSλTDQ

Pour calculer w_{k+1}, nous avons vu que LSPI pouvait utiliser deux méthodes standards, la méthode de projection du point fixe (LSTDQ) ou la méthode de minimisation du résidu quadratique (LSBRQ). Ces méthodes sont décrites par exemple par Schoknecht (2002); Munos (2003); Lagoudakis et Parr (2003). Nous les généralisons ici au cas de LSλPI.

Comme $M_k \widehat{Q}_{k+1}$ n'est pas dans l'espace défini par les fonctions de base en général, le principe de la méthode du point fixe (LSλTD) est de lui appliquer une projection orthogonale. On cherche donc la fonction de valeur approximative $\widehat{Q}_{k+1} = \Phi w_{k+1}$ qui vérifie

$$\widehat{Q}_{k+1} = \Pi_D M_k \widehat{Q}_{k+1} \tag{4.2}$$

où Π_D est une projection orthogonale sur l'espace d'approximation, définie par $\Pi = \Phi(\Phi^\mathsf{T} D \Phi)^{-1} \Phi^\mathsf{T} D$. $D = D_\mu$ représente la matrice diagonale de taille $|\mathcal{S}||\mathcal{A}|$ dont les termes sont les poids de la projection,

notés $\mu(s,a)$, où μ est une distribution de probabilités sur $S \times A$. En développant l'équation (4.2) et en utilisant de la définition de M_k (équation (4.1)), on obtient

$$\begin{aligned}
\Phi w_{k+1} &= \Phi(\Phi^\mathsf{T} D_\mu \Phi)^{-1} \Phi^\mathsf{T} D_\mu [R + (1-\lambda)\gamma P_{\pi_{k+1}} \Phi w_k + \lambda\gamma P_{\pi_{k+1}} \Phi w_{k+1}] \\
\Phi^\mathsf{T} D_\mu \Phi w_{k+1} &= \Phi^\mathsf{T} D_\mu [R + (1-\lambda)\gamma P_{\pi_{k+1}} \Phi w_k + \lambda\gamma P_{\pi_{k+1}} \Phi w_{k+1}] \\
0 &= \Phi^\mathsf{T} D_\mu [R + (1-\lambda)\gamma P_{\pi_{k+1}} \Phi w_k + \lambda\gamma P_{\pi_{k+1}} \Phi w_{k+1} - \Phi w_{k+1}].
\end{aligned}$$

Ainsi, w_{k+1} est la solution du système linéaire $Aw = b$, de taille $p \times p$ (rappelons que p est le nombre de fonctions de base), avec

$$A = \Phi^\mathsf{T} D_\mu (\Phi - \lambda\gamma P_{\pi_{k+1}} \Phi) \qquad \text{et} \qquad b = \Phi^\mathsf{T} D_\mu (R + (1-\lambda)\gamma P_{\pi_{k+1}} \Phi w_k).$$

Lorsque le nombre d'états est élevé, A et b ne peuvent pas être calculés directement, même si un modèle du PDM est disponible. Cependant, en développant la structure de A et b, on remarque que ceux-ci peuvent être exprimés sous la forme d'une espérance :

$$\begin{aligned}
A &= \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \mu(s,a)\phi(s,a) \sum_{s'\in\mathcal{S}} P(s,a,s') \Big(\phi(s,a) - \lambda\gamma\phi(s',\pi_{k+1}(s'))\Big)^\mathsf{T} \\
&= E_{(s,a)\sim\mu, s'\sim P(s,a,\cdot)} \Big[\phi(s,a)\Big(\phi(s,a) - \lambda\gamma\phi(s',\pi_{k+1}(s'))\Big)^\mathsf{T}\Big], \\
b &= \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \mu(s,a)\phi(s,a) \sum_{s'\in\mathcal{S}} P(s,a,s') \Big(R(s,a,s') + (1-\lambda)\gamma\phi(s',\pi_{k+1}(s'))^\mathsf{T} w_k\Big) \\
&= E_{(s,a)\sim\mu, s'\sim P(s,a,\cdot), r'=R(s,a,s')} \Big[\phi(s,a)\Big(r' + (1-\lambda)\gamma\phi(s',\pi_{k+1}(s'))^\mathsf{T} w_k\Big)\Big].
\end{aligned}$$

On peut alors les estimer à partir d'un ensemble de L échantillons de la forme (s,a,r',s'), avec $(s,a) \sim \mu$, $s' \sim P(s,a,\cdot)$ et $r' = R(s,a,s')$. Afin de simplifier l'écriture des estimations, nous allons en fait estimer LA et Lb, ce qui ne changera pas la solution trouvée étant donné que l'on souhaite résoudre le système linéaire $Aw = b$. Notons $\widetilde{A}$ et $\widetilde{b}$ les estimations de LA et Lb basées sur les échantillons. Pour chaque échantillon (s,a,r',s') considéré, on met à jour $\widetilde{A}$ et $\widetilde{b}$ avec

$$\begin{aligned}
\widetilde{A} &\leftarrow \widetilde{A} + \phi(s,a)\Big(\phi(s,a) - \lambda\gamma\phi(s',\pi_{k+1}(s'))\Big)^\mathsf{T}, \\
\widetilde{b} &\leftarrow \widetilde{b} + \phi(s,a)\Big(r' + (1-\lambda)\gamma\phi(s',\pi_{k+1}(s'))^\mathsf{T} w_k\Big).
\end{aligned}$$

Si la distribution des échantillons correspond à μ, alors $\widetilde{A}$ et $\widetilde{b}$ sont bien des estimateurs non biaisés de A et b. Après avoir ainsi estimé LA et Lb à partir d'une source d'échantillons, on résout le système $\widetilde{A} w_{k+1} = \widetilde{b}$ pour calculer le vecteur de paramètres w_{k+1} qui caractérise la fonction de valeur $\widehat{Q}_{k+1}$.

L'algorithme 18 (LSλTDQ) résume cette méthode de projection du point fixe. On peut facilement vérifier que, lorsque $\lambda = 1$, on retrouve bien LSTDQ. Comme dans LSTDQ, si un modèle du PDM est disponible, on peut exploiter cette connaissance. Les échantillons se résument alors à des couples états-actions (s,a) et la mise à jour de $\widetilde{A}$ et $\widetilde{b}$ devient

$$\begin{aligned}
\widetilde{A} &\leftarrow \widetilde{A} + \phi(s,a)\Big(\phi(s,a) - \lambda\gamma \sum_{s'\in\mathcal{S}} P(s,a,s')\phi(s',\pi_{k+1}(s'))\Big)^\mathsf{T}, \\
\widetilde{b} &\leftarrow \widetilde{b} + \phi(s,a) \sum_{s'\in\mathcal{S}} P(s,a,s')\Big(R(s,a,s') + (1-\lambda)\gamma\phi(s',\pi_{k+1}(s'))^\mathsf{T} w_k\Big).
\end{aligned}$$

Algorithme 18 LSλTDQ

π : politique à évaluer
$\widetilde{A} \leftarrow 0, \widetilde{b} \leftarrow 0$
pour chaque échantillon (s, a, r', s') **faire**
$\widetilde{A} \leftarrow \widetilde{A} + \phi(s,a)\Big(\phi(s,a) - \lambda\gamma\phi(s', \pi(s'))\Big)^{\mathsf{T}}$
$\widetilde{b} \leftarrow \widetilde{b} + \phi(s,a)\Big(r' + (1-\lambda)\gamma\phi(s', \pi(s'))\Big)^{\mathsf{T}} w_k$
fin pour
$w \leftarrow \widetilde{A}^{-1}\widetilde{b}$
retourner w

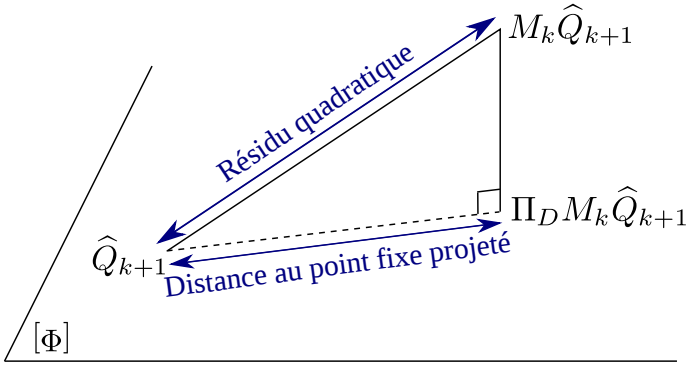


FIGURE 4.1 – Représentation schématique des deux méthodes. LSλTDQ cherche la fonction de valeur approchée qui est le point fixe de M_k suivi d'une projection sur l'espace des fonctions de valeur approximatives, alors que LSλBRQ cherche la fonction de valeur qui minimise la distance entre elle-même et une application de l'opérateur M_k.

4.1.3 Méthode de minimisation du résidu quadratique : LSλBRQ

Pour calculer la fonction de valeur approximative $\widehat{Q}_{k+1} = \Phi w_{k+1}$, on peut également proposer la méthode de minimisation du résidu quadratique, généralisée au cas $\lambda \leq 1$. Considérons l'équation (généralisée) de Bellman $Q_{k+1} = M_k Q_{k+1}$ et le résidu de Bellman défini par

$$\widehat{Q}_{k+1} - M_k \widehat{Q}_{k+1}.$$

On cherche à minimiser la norme quadratique de cette quantité, pondérée là aussi par une distribution μ :

$$\|\widehat{Q}_{k+1} - M_k \widehat{Q}_{k+1}\|_{\mu,2}.$$

On cherche donc un vecteur w_{k+1} qui minimise

$$\begin{aligned}
& \|\Phi w_{k+1} - (1-\lambda) T_{\pi_{k+1}} \Phi w_k - \lambda T_{\pi_{k+1}} \Phi w_{k+1}\|_{\mu,2} \\
= \; & \|\Phi w_{k+1} - (1-\lambda)(R + \gamma P_{\pi_{k+1}} \Phi w_k) - \lambda (R + \gamma P_{\pi_{k+1}} \Phi w_{k+1})\|_{\mu,2} \\
= \; & \|\Phi w_{k+1} - R - (1-\lambda)\gamma P_{\pi_{k+1}} \Phi w_k - \lambda \gamma P_{\pi_{k+1}} \Phi w_{k+1}\|_{\mu,2} \\
= \; & \|(\Phi - \lambda \gamma P_{\pi_{k+1}} \Phi) w_{k+1} - R - (1-\lambda)\gamma P_{\pi_{k+1}} \Phi w_k\|_{\mu,2} \\
= \; & \|\Psi w_{k+1} - c\|_{\mu,2}
\end{aligned}$$

où $\Psi = \Phi - \lambda \gamma P_{\pi_{k+1}} \Phi$ et $c = R + (1-\lambda)\gamma P_{\pi_{k+1}} \Phi w_k$. Ainsi, par une résolution standard aux moindres carrés, le vecteur de paramètres w_{k+1} qui minimise le résidu quadratique vérifie $(\Psi^\mathsf{T} D_\mu \Psi) w_{k+1} = \Psi^\mathsf{T} D_\mu c$. Notons $A = \Psi^\mathsf{T} D_\mu \Psi$ et $b = \Psi^\mathsf{T} D_\mu c$. Le problème revient alors à résoudre le système linéaire $Aw = b$, de taille $p \times p$, avec

$$\begin{aligned}
A &= (\Phi - \lambda \gamma P_{\pi_{k+1}} \Phi)^\mathsf{T} D_\mu (\Phi - \lambda \gamma P_{\pi_{k+1}} \Phi), \\
b &= (\Phi - \lambda \gamma P_{\pi_{k+1}} \Phi)^\mathsf{T} D_\mu (R + (1-\lambda)\gamma P_{\pi_{k+1}} \Phi w_k).
\end{aligned}$$

De manière analogue à LSλTD, la matrice A et le vecteur b peuvent s'écrire sous la forme d'une espérance :

$$\begin{aligned}
A = & \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \mu(s,a) \sum_{s' \in \mathcal{S}} P(s,a,s') \\
& \sum_{s'' \in \mathcal{S}} P(s,a,s'') \Big(\phi(s,a) - \lambda\gamma\phi(s'', \pi_{k+1}(s''))\Big) \Big(\phi(s,a) - \lambda\gamma\phi(s', \pi_{k+1}(s'))\Big)^\mathsf{T} \\
= \; & E_{(s,a)\sim\mu, s'\sim P(s,a,\cdot), s''\sim P(s,a,\cdot)} \\
& \left[\Big(\phi(s,a) - \lambda\gamma\phi(s'', \pi_{k+1}(s''))\Big) \Big(\phi(s,a) - \lambda\gamma\phi(s', \pi_{k+1}(s'))\Big)^\mathsf{T}\right], \\
b = & \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \mu(s,a) \sum_{s' \in \mathcal{S}} P(s,a,s') \sum_{s'' \in \mathcal{S}} P(s,a,s'') \\
& \Big(\phi(s,a) - \lambda\gamma\phi(s'', \pi_{k+1}(s''))\Big) \Big(R(s,a,s') + (1-\lambda)\gamma\phi^\mathsf{T}(s', \pi_{k+1}(s'))w_k\Big) \\
= \; & E_{(s,a)\sim\mu, s'\sim P(s,a,\cdot), r'=R(s,a,s'), s''\sim P(s,a,\cdot)} \\
& \left[\Big(\phi(s,a) - \lambda\gamma\phi(s'', \pi_{k+1}(s''))\Big) \Big(r' + (1-\lambda)\gamma\phi^\mathsf{T}(s', \pi_{k+1}(s'))w_k\Big)\right].
\end{aligned}$$

On peut donc estimer la matrice A et le vecteur b à partir d'échantillons dont la distribution correspond à μ. Comme l'espérance d'un produit est en général différente du produit des espérances, on constate ici qu'il faut, de manière analogue à LSTD et LSPI (voir par exemple les travaux de Sutton et Barto (1998); Munos (2003); Lagoudakis et Parr (2003)), utiliser pour chaque état s deux successeurs s' et s'' indépendants pour que l'estimation ne soit pas biaisée.

Notons chaque échantillon (s, a, r', s', s''), où (s', r') et s'' sont les résultats de deux réalisations indépendantes de l'action a depuis l'état s (la récompense obtenue avec l'état s'' n'est pas nécessaire). Là aussi, on note $\widetilde{A}$ et $\widetilde{b}$ les estimations de LA et Lb respectivement, où L désigne le nombre d'échantillons. Pour chaque échantillon (s, a, r', s', s''), on met à jour les estimations $\widetilde{A}$ et $\widetilde{b}$ comme suit :

$$\widetilde{A} \leftarrow \widetilde{A} + \Big(\phi(s,a) - \lambda\gamma\phi(s'', \pi_{k+1}(s''))\Big)\Big(\phi(s,a) - \lambda\gamma\phi(s', \pi_{k+1}(s'))\Big)^\mathsf{T},$$
$$\widetilde{b} \leftarrow \widetilde{b} + \Big(\phi(s,a) - \lambda\gamma\phi(s'', \pi_{k+1}(s''))\Big)\Big(r' + (1-\lambda)\gamma\phi(s', \pi_{k+1}(s'))^\mathsf{T} w_k\Big).$$

LSλBRQ est décrit dans l'algorithme 19.

Algorithme 19 LSλBRQ

π : politique à évaluer
$\widetilde{A} \leftarrow 0, \widetilde{b} \leftarrow 0$
pour chaque échantillon (s, a, r', s', s'') **faire**
  $\widetilde{A} \leftarrow \widetilde{A} + \Big(\phi(s,a) - \lambda\gamma\phi(s'', \pi(s''))\Big)\Big(\phi(s,a) - \lambda\gamma\phi(s', \pi(s'))\Big)^\mathsf{T}$
  $\widetilde{b} \leftarrow \widetilde{b} + \Big(\phi(s,a) - \lambda\gamma\phi(s'', \pi(s''))\Big)\Big(r' + (1-\lambda)\gamma\phi(s', \pi(s'))^\mathsf{T} w_k\Big)$
fin pour
$w \leftarrow \widetilde{A}^{-1}\widetilde{b}$
retourner w

Enfin, si l'on dispose d'un modèle du PDM, les échantillons peuvent se limiter à des couples états-actions (s, a) et la mise à jour des estimations devient

$$\widetilde{A} \leftarrow \widetilde{A} + \Big(\phi(s,a) - \lambda\gamma \sum_{s'' \in \mathcal{S}} P(s,a,s'')\phi(s'', \pi_{k+1}(s''))\Big)$$
$$\Big(\phi(s,a) - \lambda\gamma \sum_{s' \in \mathcal{S}} P(s,a,s')\phi(s', \pi_{k+1}(s'))\Big)^\mathsf{T},$$
$$\widetilde{b} \leftarrow \widetilde{b} + \Big(\phi(s,a) - \lambda\gamma \sum_{s'' \in \mathcal{S}} P(s,a,s'')\phi(s'', \pi_{k+1}(s''))\Big)$$
$$\sum_{s' \in \mathcal{S}} P(s,a,s')\Big(R(s,a,s') + (1-\lambda)\gamma\phi(s', \pi_{k+1}(s'))^\mathsf{T} w_k\Big).$$

On notera que, lorsqu'un modèle du PDM est disponible, comme les échantillons se limitent à des couples états-actions (s, a), la contrainte de devoir générer les états et récompensenses suivants en double disparaît. Le reste de l'algorithme est identique au cas de LSλTD : une fois LA et Lb estimés, on résout le système linéaire $\widetilde{A}w = \widetilde{b}$ afin d'obtenir la fonction de valeur $\widehat{Q}_{k+1}$.

Que ce soit avec LSλTD ou LSλBR, il est possible d'utiliser la formule de Sherman-Morrison pour maintenir à jour directement une estimation de A^{-1} et ainsi réduire la complexité du calcul de $\widetilde{A}^{-1}\widetilde{b}$. La mise à jour de $\widetilde{A}$ est alors remplacée par

$$\widetilde{A}^{-1} \leftarrow \widetilde{A}^{-1} + \frac{\widetilde{A}^{-1} u v^\mathsf{T} \widetilde{A}^{-1}}{1 + v^\mathsf{T} \widetilde{A}^{-1} u} \tag{4.3}$$

avec, dans le cas de LSλTDQ,

$$u = \phi(s,a) \text{ et } v = \phi(s,a) - \lambda\gamma\phi(s', \pi(s')),$$

et, dans le cas de LSλBRQ,

$$u = \phi(s,a) - \lambda\gamma\phi(s'', \pi(s'')) \text{ et } v = \phi(s,a) - \gamma\phi(s', \pi(s')).$$

Synthèse des méthodes d'évaluation d'une politique π_{k+1}					
Algorithme d'évaluation	Critère minimisé pour évaluer π_{k+1}	Évaluation optimiste	Compromis biais-variance (λ)	Off-policy	Échantillonnage efficace
TD(0)	$\widehat{V} = \Pi_{D_\pi} T_0 \widehat{V}$				
TD(λ)	$\widehat{V} = \Pi_{D_\pi} T_\lambda \widehat{V}$		×		
LSTD(λ)			×		×
LSPE(λ)			×		×
AλPI (phase d'évaluation)	$\widehat{V} = \Pi_{D_\pi} T_\lambda \widehat{V}_k$	×	×		×
LSTDQ	$\widehat{Q} = \Pi_D T_0 \widehat{Q}$			×	×
LSλTDQ	$\widehat{Q} = \Pi_D M_k \widehat{Q}$	×	×	×	×
LSBRQ	$\|\widehat{Q} - T_0 \widehat{Q}\|^2$			×	×
LSλBRQ	$\|\widehat{Q} - M_k \widehat{Q}\|^2$	×	×	×	×

TABLE 4.1 – Positionnement des deux méthodes d'évaluation de LSλPI (LSλTDQ et LSλBRQ) par rapport aux autres méthodes d'évaluation de la fonction de valeur. Chaque méthode est ici vue comme un choix possible pour la phase d'évaluation de la politique dans un schéma d'itération sur les politiques à l'itération k. La politique à évaluer, notée π_{k+1}, est la politique gloutonne par rapport à la valeur précédente $\widehat{V}_k$ ou $\widehat{Q}_k$. LSλPI peut être vu comme une implémentation off-policy de λPI, comme une généralisation optimiste de LSPI avec un paramètre λ, ou encore comme une version avec contrôle off-policy de LSPE(λ).

4.1.4 Least-Squares λ Policy Iteration

Au final, LSλPI est résumé dans l'algorithme 20, pour le choix d'une méthode (LSλTDQ ou LSλBRQ) et d'une règle de mise à jour des estimations (avec ou sans modèle).

Algorithme 20 LSλPI

$\mathcal{D} \leftarrow$ ensemble d'échantillons de la forme $\begin{cases} (s,a,r',s') & \text{avec LS}\lambda\text{TDQ} \\ (s,a,r',s',s'') & \text{avec LS}\lambda\text{BRQ} \end{cases}$
$k \leftarrow 0, w_0 \leftarrow$ initialisation arbitraire
répéter
  $\pi_{k+1} \leftarrow \text{glouton}(\Phi w_k)$
  $w_{k+1} \leftarrow \begin{cases} \text{LS}\lambda\text{TDQ}(\mathcal{D}, \pi_{k+1}) \\ \text{ou} \\ \text{LS}\lambda\text{BRQ}(\mathcal{D}, \pi_{k+1}) \end{cases}$
  $k \leftarrow k+1$
jusqu'à $\|w_k - w_{k-1}\|_\infty < \epsilon$

LSλPI est un algorithme utilisant le paramètre λ, qui évalue les politiques avec une méthode du second ordre, et qui itère sur les politiques. Dans la littérature, les travaux aux moindres carrés qui évaluent une politique fixée, comme LSTD(λ) (Boyan, 2002) et LSPE(λ), pourraient aussi être utilisés dans un contexte d'itération sur les politiques afin de traiter des problèmes de contrôle. La principale différence de LSλPI avec l'état de l'art est qu'il s'agit d'un algorithme optimiste : il ne nécessite pas d'estimer la valeur de la politique gourmande, mais seulement de suivre sa direction avec un pas ajustable selon la valeur de λ. L'illustration de la figure 1.2 (page 28) est toujours valide dans le cas des méthodes du second ordre : LSλPI peut être vu comme une version optimiste de LSPI où l'on change de politique avant que la phase d'évaluation de la politique courante soit terminée. Par ailleurs, on peut considérer un algorithme d'itération sur les politiques qui se baserait sur LSPE(λ) pour évaluer chaque politique. Aux détails d'approximation stochastique près, alors que l'évaluation via LSPE(λ) consiste à appliquer l'opérateur T_λ une infinité de fois pour évaluer la politique, LSλPI l'applique seulement une fois et change de politique ensuite.

En ce sens, LSλPI est proche de l'algorithme AλPI (Bertsekas et Ioffe, 1996), présenté à la section 3.4.3, où l'on applique une seule fois T_λ (de façon approximative) pour changer de politique ensuite. Cependant, AλPI est on-policy étant donné qu'il se fonde sur les mêmes mécanismes que LSPE(λ). En effet, l'application approximative de l'opérateur T_λ s'appuie sur l'équation $T_\lambda V_k = V_k + \Delta_k$ avec LSPE(λ) et AλPI, alors qu'avec LSλPI, elle s'appuie sur l'équation $T_\lambda V_k = M_k^\infty V_k$. Notre approche permet, comme avec LSPI, d'évaluer les politiques de façon off-policy. Tous ces algorithmes sont récapitulés dans le tableau 4.1.

Revenons plus en détail sur les deux méthodes permettant de mettre à jour la fonction de valeur dans LSλPI : la méthode de projection du point fixe (LSλTDQ) et la méthode de minimisation du résidu quadratique (LSλBRQ). Ces deux méthodes sont deux moyens de calculer $\widehat{Q}_{k+1}$ en cherchant à minimiser des critères différents, et les solutions qu'elles trouvent sont en général différentes. Un cas où les deux approches sont équivalentes est lorsque $\lambda = 0$: en effet, on peut voir que les estimations $\widetilde{A}$ et $\widetilde{b}$ sont alors construites de la même manière. L'algorithme revient dans ce cas à effectuer Fitted Value Iteration avec régression linéaire, une version approximative de Value Iteration (Szepesvári et Munos, 2005). Dans le cas particulier où $\lambda = 1$, ce qui correspond aux évaluations faites par LSPI, LSTDQ semble donner de meilleurs résultats (Lagoudakis et Parr, 2003). De plus, sur certains exemples, on peut montrer que LSBRQ ne calcule pas la bonne solution alors que LSTDQ le fait (Sutton *et al.*, 2009). Cependant, Schoknecht (2002) a montré que LSTDQ est moins stable numériquement. En effet, la matrice A correspondant à LSTDQ peut être singulière.

4.1.5 Cas possible d'une erreur non contrôlée

Discutons maintenant de la convergence de LSλPI. Notons que comme LSλPI est une version approximative de λPI, la garantie de performance du théorème 1 (page 44) s'applique. En effet, nous avons vu au chapitre 2 que λPI est un cas particulier de Unified Policy Iteration. Cette garantie de performance s'applique lorsque l'erreur d'approximation ϵ_k est bornée à chaque itération. Elle établit que sous cette condition, la distance de la valeur de politique obtenue par rapport à la valeur optimale est bornée. À l'inverse, si l'erreur d'approximation n'est pas contrôlée et augmente à chaque itération, il se peut que l'algorithme diverge. Nous étudions ici un cas où l'algorithme peut diverger selon les valeurs de λ et γ.

Nous nous intéressons à un exemple simple tiré de Bertsekas et Tsitsiklis (1996, page 334), sur lequel les auteurs montrent, dans le cas de Fitted Value Iteration ($\lambda = 0$), que l'estimation de la fonction de valeur peut diverger pour certaines valeurs de γ, alors même que la capacité d'approximation de l'architecture linéaire permet de représenter exactement la fonction de valeur cible. Nous nous intéressons dans ce qui suit à la convergence des deux méthodes d'évaluation que LSλPI peut employer (LSλTDQ et LSλBRQ), et nous considérons les autres valeurs de λ (rappelons que, lorsque $\lambda = 0$, les deux méthodes sont équivalentes).

On considère un système non contrôlé avec 2 états, de sorte que les fonctions de valeurs sont définies uniquement sur l'espace d'états. La matrice de transition est donnée par

$$P = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}.$$

L'état 2 est absorbant et les récompenses sont 0. On a donc $Q(1) = Q(2) = 0$. On considère un approximateur linéaire avec $\Phi = (1\ 2)^\mathsf{T}$. Ici, la valeur peut être représentée exactement par l'espace choisi.

Méthode du point fixe projeté (LSλTDQ)

Dans le cas de LSλTDQ, l'évolution des poids est régie par l'équation

$$w_{k+1} = (I - \lambda\gamma(\Phi^\mathsf{T}\Phi)^{-1}\Phi^\mathsf{T}P\Phi)^{-1}(1-\lambda)\gamma(\Phi^\mathsf{T}\Phi)^{-1}\Phi^\mathsf{T}P\Phi w_k.$$

On suppose ici que les échantillons sont distribués uniformément, c'est-à-dire $D_\mu = I$. On a $\Phi^\mathsf{T}\Phi = 5$ et donc $(\Phi^\mathsf{T}\Phi)^{-1}\Phi^\mathsf{T} = (1/5\ 2/5)$. Comme $P\Phi = (2\ 2)^\mathsf{T}$, on en déduit que $(\Phi^\mathsf{T}\Phi)^{-1}\Phi^\mathsf{T}P\Phi$ vaut 6/5. Autrement dit, on a

$$w_{k+1} = \alpha_{\mathrm{TD}} w_k$$

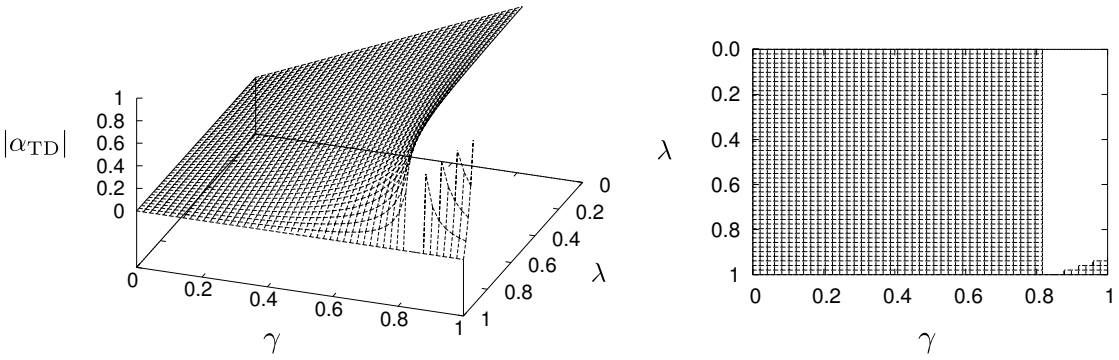


FIGURE 4.2 – Méthode LSλTDQ. Gauche : $|\alpha_{\mathrm{TD}}|$ en fonction de λ et γ. Droite : domaine où $|\alpha_{\mathrm{TD}}| < 1$.

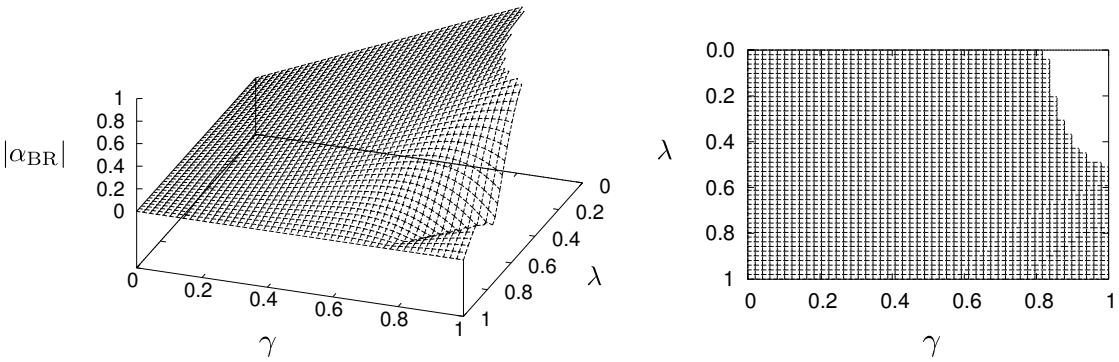


FIGURE 4.3 – Méthode LSλBRQ. Gauche : $|\alpha_{\mathrm{BR}}|$ en fonction de λ et γ. Droite : domaine où $|\alpha_{\mathrm{BR}}| < 1$.

avec

$$\alpha_{\mathrm{TD}} = \frac{(1-\lambda)\frac{6}{5}\gamma}{1-\lambda\frac{6}{5}\gamma}.$$

L'algorithme converge vers la solution si et seulement si $|\alpha_{\mathrm{TD}}| < 1$. Ce coefficient admet des singularités : pour $\lambda\gamma$ proche de $5/6$, il tend vers $\pm\infty$. La figure 4.2 donne une représentation graphique de la valeur absolue de ce coefficient en fonction de λ et γ, ainsi que le domaine où ce coefficient a un module inférieur à 1.

Méthode du résidu quadratique (LSλBRQ)

Avec LSλBRQ, l'évolution des poids est régie par l'équation

$$w_{k+1} = (\Psi^{\mathsf{T}}\Psi)^{-1}\Psi^{\mathsf{T}}(1-\lambda)\gamma P\Phi w_k$$

avec $\Psi = \Phi - \lambda\gamma P\Phi = (1-2\lambda\gamma, 2-2\lambda\gamma)^{\mathsf{T}}$. Ainsi $\Psi\Psi^{\mathsf{T}} = (1-2\lambda\gamma)^2 + (2-2\lambda\gamma)^2$, quantité qui est toujours strictement supérieure à $1/2$. Comme $P\Phi = (2\ 2)^{\mathsf{T}}$, on en déduit que $\Psi^{\mathsf{T}}P\Phi = 6 - 8\lambda\gamma$. Au final, on a

$$w_{k+1} = \alpha_{\mathrm{BR}} w_k$$

avec

$$\alpha_{\mathrm{BR}} = \frac{(1-\lambda)\gamma(6-8\lambda\gamma)}{(1-2\lambda\gamma)^2 + (2-2\lambda\gamma)^2}.$$

La figure 4.3 donne comme précédemment une représentation graphique de la valeur absolue de ce coefficient en fonction de λ et γ, ainsi que le domaine où ce coefficient a une valeur absolue inférieure à 1.

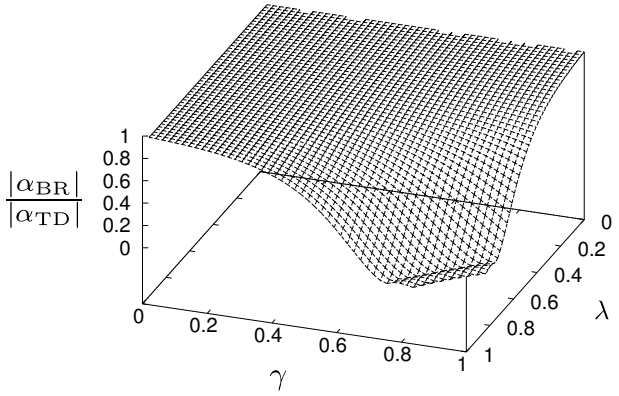


FIGURE 4.4 – $\frac{|\alpha_{\mathrm{BR}}|}{|\alpha_{\mathrm{TD}}|}$ en fonction de λ et γ.

Comparaison

On observe que, sur l'exemple étudié, la région où LSλBRQ converge est strictement plus grande que celle de LSλTDQ. En particulier, pour toute valeur de γ, le choix parmi les valeurs de λ est plus grand pour LSλBRQ que pour LSλTDQ. On notera également que, pour la valeur limite $\gamma = 5/6$, LSλTDQ ne converge que si l'on prend $\lambda = 1$. Enfin, dans le cas où les deux méthodes convergent, on peut voir sur le graphique de la figure 4.4, où nous avons tracé la courbe $\frac{|\alpha_{\mathrm{BR}}|}{|\alpha_{\mathrm{TD}}|}$ en fonction de λ et γ, que LSλBRQ converge toujours plus vite que LSλTDQ.

4.2 Expériences

Nous avons introduit l'algorithme LSλPI, qui ajoute à LSPI (Lagoudakis et Parr, 2003) le caractère optimiste et la possibilité de faire un compromis biais-variance de λPI (Bertsekas et Ioffe, 1996), et nous avons discuté de sa validité de manière théorique en donnant une garantie de convergence sous certaines conditions (théorème 1). Nous venons également de voir que, dans le cas de l'évaluation d'une politique fixée, le choix de λ peut influer sur la convergence ou non de l'algorithme.

Nous présentons maintenant quelques expériences sur un problème d'itération sur les politiques afin de montrer l'intérêt de LSλPI d'un point de vue expérimental. Etant donné que LSλPI est une généralisation de LSPI, nous avons réalisé des expériences sur un problème d'optimisation de politique précédemment étudié par Lagoudakis et Parr (2003) dans le cadre de LSPI. Il s'agit d'un PDM de type chaîne d'états dans lequel on connaît la fonction de valeur optimale exacte, de sorte à pouvoir facilement évaluer les performances obtenues. Dans le chapitre 6, nous donnerons également des résultats sur le jeu de Tetris, qui est un problème plus difficile et à grand espace d'états et qui a également été traité par Lagoudakis et Parr (2003) avec LSPI.

La figure 4.5 représente le PDM simple considéré par Lagoudakis et Parr (2003) pour illustrer le comportement de LSPI. Il s'agit d'une chaîne de 20 états avec deux actions possibles : gauche (L) ou droite (R). Chaque action envoie dans la bonne direction avec une probabilité de $0,9$, et dans la direction opposée avec une probabilité de $0,1$. Lorsque l'agent arrive à un des deux états aux extrémités de la chaîne, il obtient une récompense de 1. Dans tous les autres états, il obtient une récompense nulle. Il est clair que la politique optimale est celle qui consiste à aller vers la gauche depuis les états situés dans la moitié gauche de la chaîne, et vers la droite depuis les autres. La fonction de valeur optimale peut être calculée facilement et de manière exacte. Ainsi, lors de nos expériences, on pourra tracer la courbe représentant la distance entre la valeur courante et la valeur optimale (pour mesurer la qualité de l'approximation), et la distance entre la valeur de la politique courante et la valeur optimale (pour mesurer la qualité de la politique obtenue).

Dans ces expériences, on n'utilisera pas la connaissance du modèle du PDM (transitions et récom-

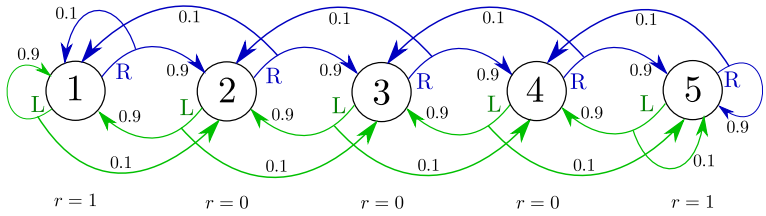


FIGURE 4.5 – Le PDM étudié, représenté ici avec 5 états (nos expériences comportent 20 états). Chaque action (L ou R) envoie dans la bonne direction avec une probabilité de $0,9$ et dans la direction opposée avec une probabilité de $0,1$. Les deux extrémités comportent une récompense de 1.

penses). Comme Lagoudakis et Parr (2003), nous avons testé deux jeux de fonctions de base pour représenter l'espace d'états. Le premier est un ensemble de fonctions de base polynômiales répétées pour chacune des deux actions :

$$\phi(s,a) = \begin{pmatrix} \mathbb{1}_{a=L} \times 1 \\ \mathbb{1}_{a=L} \times s \\ \mathbb{1}_{a=L} \times s^2 \\ \mathbb{1}_{a=R} \times 1 \\ \mathbb{1}_{a=R} \times s \\ \mathbb{1}_{a=R} \times s^2 \end{pmatrix}$$

où s est le numéro d'état (de 1 à 20), et $\mathbb{1}_{a=X} = 1$ si $a = X$ et 0 sinon. Le second jeu de fonctions est un ensemble de gaussiennes dont les moyennes sont distribuées uniformément sur l'espace d'états et dont la variance est définie par $\sigma = 4$. Pour chaque action, on a 10 gaussiennes et un terme constant, ce qui donne un total de 22 fonctions de base.

Influence de λ

Nous avons observé que la convergence de la fonction de valeur est plus difficile lorsque le nombre d'échantillons est faible, ou lorsque γ est élevé (c'est-à-dire lorsque l'horizon du problème est grand). Si le nombre d'échantillons est suffisamment important ou si γ est peu élevé, λ n'a que peu d'influence car la variance de l'estimation discutée précédemment pose moins de problèmes. Il est alors préférable d'utiliser $\lambda = 1$ afin de converger plus rapidement. Dans les cas de convergence plus difficile, on observe plus clairement une influence du paramètre λ. La figure 4.6 représente la distance entre la fonction de valeur à chaque itération et la fonction de valeur optimale, moyennée sur 10 exécutions ayant des ensembles d'échantillons différents, et ce pour plusieurs valeurs de λ. Les ensembles d'échantillons comportent des épisodes de 200 états visités avec la politique qui choisit une action aléatoire uniformément. La méthode utilisée est LSλTDQ dans le graphique du haut, et LSλBRQ dans le graphique du bas. Dans les deux cas, on utilise l'approximateur gaussien et $\gamma = 0.95$. Comme attendu, on observe que pour $\lambda < 1$, l'approximation est meilleure car la variance de l'estimation est réduite. En contrepartie, un plus grand nombre d'itérations est nécessaire pour atteindre cette bonne approximation. En effet, comme discuté précédemment, utiliser une valeur de λ inférieure à 1 introduit un biais dans la mesure où on ne cherche plus à s'approcher le plus possible de la valeur de la politique courante, mais seulement d'un certain pas dans sa direction. On remarquera que ces courbes sont similaires à celles de Kearns et Singh (2000) qui proposent une analyse théorique du compromis biais-variance de TD(λ).

On observe que pour $\lambda = 1$, au bout de quelques itérations seulement, la fonction de valeur cesse de s'améliorer et se met à osciller en restant relativement loin de l'optimal par rapport aux valeurs de λ inférieures. Les valeurs intermédiaires de λ offrent le meilleur compromis en donnant une bonne approximation et avec un nombre d'itérations raisonnable. Sur la plupart des expériences que nous avons effectuées, LSλTDQ et LSλBRQ donnent des performances similaires, avec un léger avantage pour LSλTDQ. On observe ainsi sur la figure 4.6 qu'avec LSλBRQ, il y a plus de valeurs de λ pour lesquelles la fonction de valeur se stabilise trop rapidement, avant d'avoir atteint une bonne approximation. En pratique, LSλTDQ semble donc un peu plus performant que LSλBRQ étant donné qu'il donne de bons résultats pour un plus

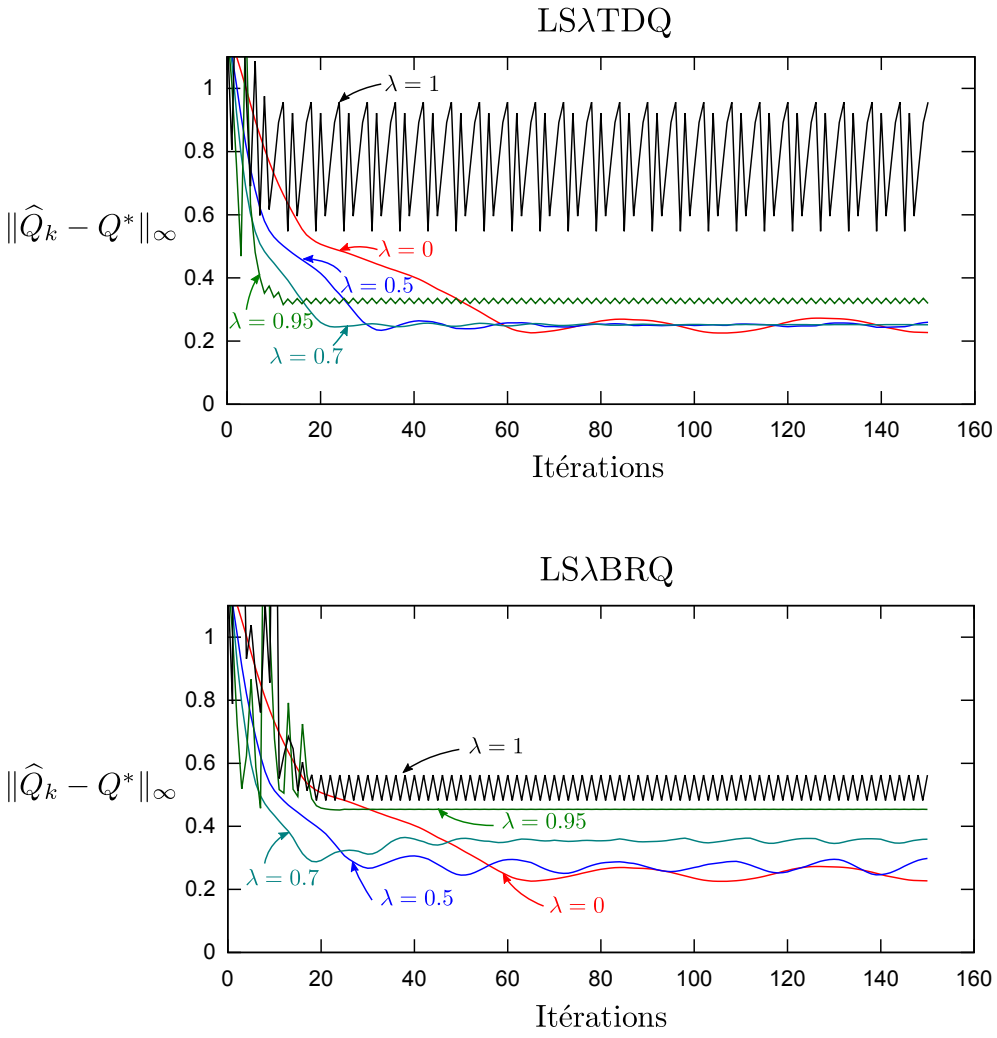


FIGURE 4.6 – Chaîne d'états : évolution au cours des itérations de $\|\widehat{Q}_k - Q^*\|_\infty$, distance entre la fonction de valeur approximative courante et la fonction de valeur optimale, pour plusieurs valeurs de λ. Fonctions de base gaussiennes. $\gamma = 0.95$. Moyenne de 10 exécutions, les exécutions utilisant des ensembles d'épisodes de 200 échantillons. Haut : méthode LSλTDQ. Bas : méthode LSλBRQ.

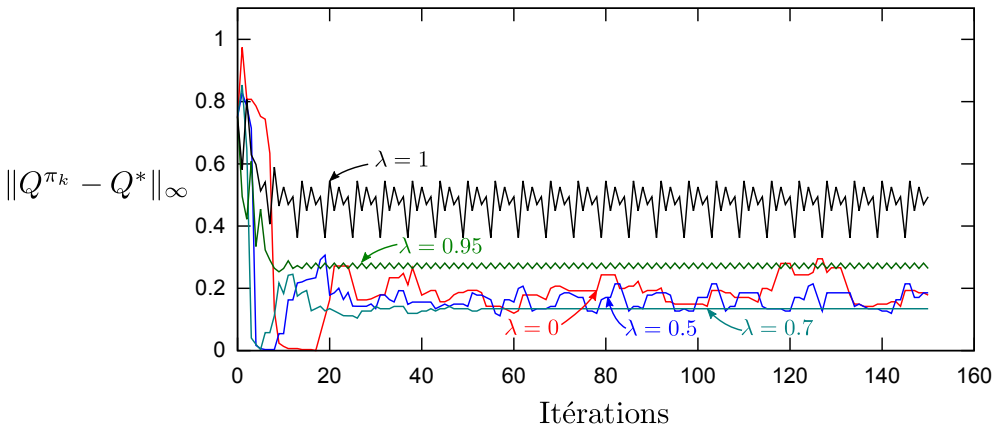


FIGURE 4.7 – Chaîne d'états : évolution de $\|Q^{\pi_k} - Q^*\|_\infty$, distance entre la valeur de la politique courante et la valeur optimale, pour l'expérience de la figure 4.6 avec LSλTDQ. On observe que la politique oscille avec une fréquence qui augmente avec λ et elle converge pour une valeur intermédiaire ($\lambda = 0,7$).

grand intervalle de valeurs de λ. Cependant, il peut en théorie poser des problèmes de stabilité numérique dans certains cas (voir section 4.1.4), bien que nous n'ayons pas rencontré de tels problèmes au cours de nos expériences.

On remarque également, surtout dans le cas de LSλBRQ, qu'il serait intéressant d'utiliser une valeur décroissante de λ. En effet, dans les premières itérations, une valeur de λ proche de 1 permet de s'approcher rapidement de la fonction de valeur optimale, puis au fur et à mesure des itérations, les valeurs de λ plus petites conduisent à une meilleure approximation.

Par ailleurs, on constate que lorsque la fonction de valeur ne converge pas, la politique oscille avec une fréquence qui augmente avec λ. Cela se produit lorsqu'il y a un cycle dans la séquence des politiques. On peut observer ce phénomène sur la figure 4.7, qui représente $\|Q^{\pi_k} - Q^*\|_\infty$ pour l'expérience de la figure 4.6 avec LSλTDQ. Pour les petites valeurs de λ, la politique oscille lentement car LSλPI réalise des petits pas. Lorsque λ augmente, les oscillations sont plus rapides puisque les pas sont plus importants (voir la vue intuitive de la figure 1.2, page 28). Il est intéressant de constater qu'il y a ensuite des valeurs intermédiaires de λ pour lesquelles la politique converge (par exemple $\lambda = 0,7$). Enfin, pour les grandes valeurs de λ, la politique ne converge plus et oscille à nouveau, avec une fréquence plus importante. La possibilité d'utiliser λ pour stabiliser la politique est d'autant plus intéressante car on peut montrer que lorsque la politique a convergé, le coefficient $\frac{2\gamma}{(1-\gamma)^2}$ du théorème 1 (page 44), qui traduit l'erreur relative de la fonction de valeur courante par rapport à la fonction de valeur optimale, est réduit d'un facteur $1 - \lambda\gamma$.

Cas d'une politique fixée

La figure 4.8 représente une expérience où la convergence est moins difficile que sur l'exemple précédent car $\gamma = 0,9$ (les autres paramètres sont inchangés et les échantillons sont les mêmes). On s'intéresse ici à LSλTDQ uniquement, où l'on observe un phénomène qui n'apparaît pas avec LSλBRQ. Le graphique du haut indique la qualité de la politique courante : il représente, comme sur la figure 4.7, la distance entre la valeur de la politique courante et la valeur optimale. On remarque que la politique converge, excepté dans le cas où $\lambda = 1$. Le graphique du bas représente, comme précédemment, la distance entre la fonction de valeur approximative courante et la fonction de valeur optimale. A partir de l'itération 40 environ, la politique à évaluer devient la même pour toutes les valeurs de λ pour lesquelles la politique a convergé. Il semble alors, d'après le graphique, que l'estimation de la fonction de valeur converge vers la même quantité quelle que soit la valeur de λ.

Nous pouvons en effet vérifier analytiquement que, lorsque la politique est fixée et que LSλTDQ converge, elle converge vers une valeur qui ne dépend pas de λ. Ce n'est pas le cas de LSλBRQ en

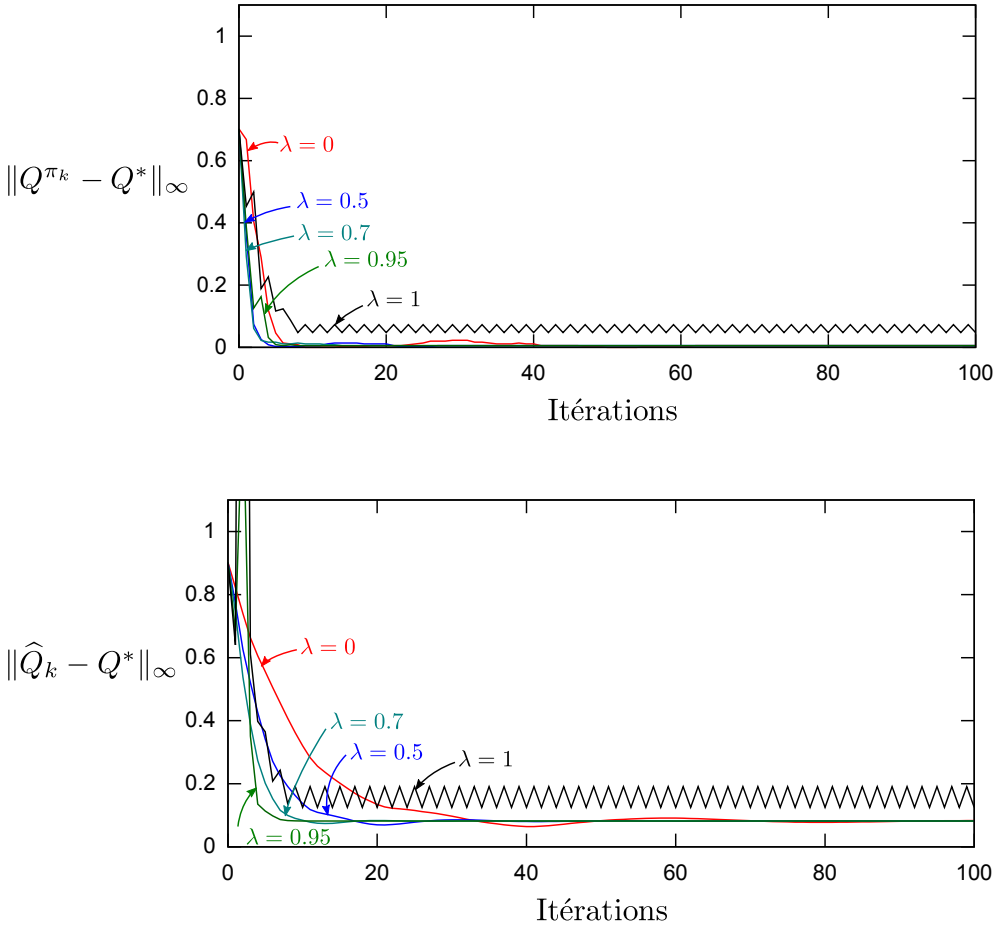


FIGURE 4.8 – Chaîne d'états : Résultat de LSλTDQ appliquée avec $\gamma = 0.9$ et des fonctions de base gaussiennes. Haut : Evolution de $\|\widehat{Q}^{\pi_k} - Q^*\|_\infty$, distance entre la fonction de valeur exacte de la politique courante, pour plusieurs valeurs de λ. Bas : Evolution de $\|\widehat{Q}_k - Q^*\|_\infty$, distance entre la fonction de valeur approximative courante et la fonction de valeur optimale, pour plusieurs valeurs de λ. Fonctions de base gaussiennes. $\gamma = 0.9$. Moyenne de 10 exécutions, chaque exécution utilisant une ensemble différent d'épisodes de 200 échantillons. On observe que lorsque la politique est la même pour différentes valeurs de λ, la fonction de valeur semble converger vers une limite qui ne dépend pas de λ. Nous avons vérifié cette propriété analytiquement, propriété qui ne s'applique pas à LSλBRQ.

général. En utilisant la définition de M_k (équation 1.5), le fait que $\widehat{Q}_{k+1} = \widehat{Q}_k$ et que $\pi_{k+1} = \pi_k$, on a

$$\widehat{Q}_{k+1} = \Pi_D M_k \widehat{Q}_{k+1} = \Pi_D((1-\lambda) T_{\pi_{k+1}} \widehat{Q}_{k+1} + \lambda T_{\pi_{k+1}} \widehat{Q}_{k+1}) = \Pi_D T_{\pi_{k+1}} \widehat{Q}_{k+1}.$$

Ainsi, $\widehat{Q}_{k+1}$ converge vers le point fixe de $\Pi_D T_{\pi_{k+1}}$, qui ne dépend pas de λ. On pourrait alors penser que λ est inutile dans LSλTDQ, mais rappelons que ce n'est qu'en cas de convergence de la politique et de la fonction de valeur que cette dernière cesse de dépendre de λ. Or, nous avons vu que c'est justement le réglage de λ qui peut permettre d'obtenir la convergence ou non. Cette propriété suggère que le choix de λ est plus difficile dans le cas de LSλBRQ étant donné qu'il influe non seulement sur la convergence, mais aussi sur la fonction de valeur obtenue après convergence de la politique.

Conclusion

Nous avons proposé dans ce chapitre LSλPI, une implémentation du second ordre de λPI (Bertsekas et Ioffe, 1996). Notre algorithme généralise LSPI (Lagoudakis et Parr, 2003) en y ajoutant l'évaluation optimiste de λPI. LSλPI est à notre connaissance le premier algorithme qui cumule les quatre caractéristiques suivantes : l'utilisation d'un paramètre λ qui règle un compromis biais-variance lors de l'estimation de la fonction de valeur, l'évaluation optimiste de la politique, l'approximation par une méthode du second ordre pour exploiter les échantillons de manière efficace et l'itération sur les politiques off-policy. Il conserve les avantages de LSPI : le modèle du PDM n'est pas requis (mais il peut être intégré s'il est connu), et il n'est pas nécessaire de générer de nouvelles trajectoires d'échantillons à chaque fois que la politique change.

Nous avons présenté des résultats expérimentaux qui confirment l'influence de λ sur la qualité de l'approximation et la performance des politiques générées. Nos résultats empiriques sur un problème simple, précédemment traité par LSPI (Lagoudakis et Parr, 2003), montrent que les valeurs de λ intermédiaires (différentes de 0 et 1) peuvent donner de meilleurs résultats en pratique lorsque le nombre d'échantillons est limité. Dans le chapitre 6, nos expériences sur le problème de Tetris confirmeront qu'un réglage intermédiaire de λ permet d'améliorer l'efficacité de l'échantillonnage. Cela peut s'avérer intéressant dans des applications d'apprentissage où le nombre d'échantillons disponibles est restreint.

Cependant, avec LSλPI, comme avec les autres algorithmes du second ordre et ceux utilisant un paramètre λ, un problème qui reste posé est de savoir quelle méthode d'estimation choisir (LSλTDQ ou LSλBRQ) et comment fixer la valeur de λ. λ peut en effet avoir une influence cruciale sur la convergence ou non de l'algorithme et sur les performances obtenues. Les expériences suggèrent qu'avec LSλTDQ, il y a une plus grande plage de valeurs de λ qui permettent d'obtenir de bonnes performances. Cela confirme la tendance selon laquelle LSλTDQ donnerait des résultats légèrement meilleurs en pratique. Concernant le choix de λ, nous remarquons qu'une valeur décroissante de λ peut offrir le meilleur compromis entre la vitesse de convergence et la qualité de l'estimation. Kearns et Singh (2000) proposent une méthode analytique pour déterminer la valeur optimale de λ à chaque itération dans le cas de TD(λ). Il serait intéressant d'étudier une approche similaire pour LSλPI.

Troisième partie

Etude de cas : le jeu de Tetris

Chapitre 5

Etat de l'art des travaux sur Tetris

Nous avons jusqu'ici étudié les aspects théoriques de l'apprentissage par renforcement pour les problèmes à grands espaces d'états. Dans cette partie, nous allons nous intéresser en détail à une application particulière qui est le jeu de Tetris. Ce problème à grand espace d'états a été considéré par de nombreux algorithmes de la littérature d'apprentissage par renforcement mais aussi par des approches d'optimisation directe. Comme nous allons le voir, si les approches d'apprentissage par renforcement permettent de calculer une politique optimale sur une taille réduite du jeu, et parviennent à traiter la taille normale du jeu avec une relative efficacité, ce sont des approches d'optimisation directe de la politique et notamment des algorithmes de type évolutionnaires qui atteignent les meilleures performances sur la taille normale du jeu. De plus, nos observations et nos résultats vont mettre en évidence le fait que dans un tel problème, la connaissance qu'un expert du domaine peut apporter aux algorithmes a une influence significative et peut parfois s'avérer encore plus décisive sur les performances que le choix de l'algorithme lui-même.

Dans ce chapitre, nous présentons le problème de Tetris et nous dressons une revue des principaux travaux qui s'y sont intéressés. Ces travaux incluent des approches d'apprentissage par renforcement, des approches d'optimisation directe et des algorithmes simplement réglés à la main. Nous développons en particulier deux approches qui se sont montrées particulièrement performantes sur Tetris : l'algorithme de Dellacherie (Fahey, 2003), un contrôleur réglé à la main mais doté d'une excellente connaissance experte, et la méthode d'entropie croisée (Szita et Lőrincz, 2006), une approche évolutionnaire d'optimisation directe de la politique.

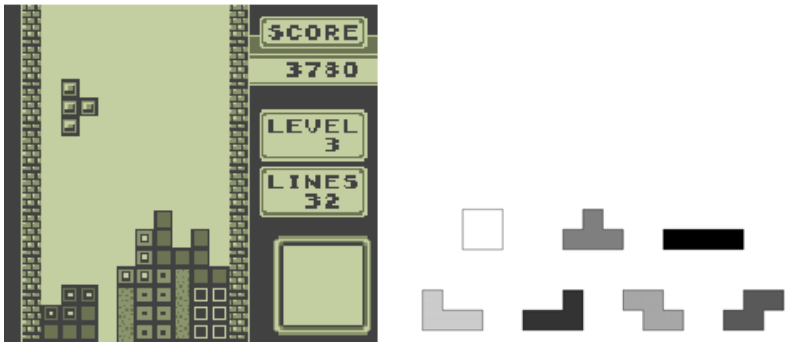
5.1 Le problème de Tetris

Tetris est un célèbre jeu vidéo créé en 1985 par Alexey Pajitnov. Le jeu se déroule sur une grille 2D de taille 10×20 (10 colonnes et 20 lignes), où des pièces de différentes formes tombent à une certaine vitesse du haut de la grille les unes après les autres (voir figure 5.1). Le joueur doit choisir où il place chaque pièce : il peut déplacer la pièce horizontalement et la faire pivoter. Lorsqu'une ligne horizontale est pleine, celle-ci est supprimée et toutes les cellules qui étaient au-dessus d'elle descendent d'une ligne. L'objectif est de supprimer un maximum de lignes avant que la partie ne soit terminée. La partie se termine lorsqu'il ne reste plus assez d'espace libre en haut de la grille pour placer la pièce courante. On peut trouver une spécification détaillée de Tetris sur le site de Fahey (2003).

Tetris a fait l'objet de plusieurs travaux de recherche (la section 5.2 en donne un aperçu). C'est un problème difficile malgré ses règles simples : il contient un grand nombre de configurations (de l'ordre de $2^{200} \simeq 10^{60}$ états [7]). De plus, on sait aujourd'hui que trouver une séquence de coups qui maximise le nombre de lignes est un problème NP-complet, même dans le cas où la séquence de pièces est connue à l'avance (Demaine *et al.*, 2003).

Pour comparer la qualité de plusieurs joueurs artificiels à Tetris, il est important de définir une mesure de performance. Comme Tetris est un jeu stochastique, une mesure naturelle, considérée par tous les travaux dont nous avons connaissance, est le nombre moyen de lignes réalisées par le joueur artificiel

7. Ce nombre est approximatif car il inclut un certain nombre de configurations impossibles (voir Fahey, 2003).



(a) (b)

FIGURE 5.1 – Une image du jeu de Tetris (a). Les sept pièces possibles (b)

avant de perdre la partie. Cette mesure est toujours finie : il a été montré que toute partie de Tetris se termine avec probabilité 1 (Burgiel, 1997).

Comme la plupart des chercheurs, nous nous intéressons à une version simplifiée de Tetris. D'abord, nous considérons uniquement les **contrôleurs à une pièce**, c'est-à-dire les joueurs artificiels qui connaissent la configuration de la grille et la pièce courante uniquement. Un contrôleur qui connaît également la forme de la prochaine pièce, comme cela est le cas dans le Tetris original, est appelé un **contrôleur à deux pièces**. Un tel contrôleur est avantagé puisqu'il peut tirer parti de cette connaissance supplémentaire pour prendre de meilleures décisions. Il est assez immédiat d'étendre des travaux sur des contrôleurs à une pièce pour construire des contrôleurs à deux pièces. Des données expérimentales suggèrent que les performances d'un contrôleur sont ainsi améliorées de plusieurs ordres de grandeur (Fahey, 2003). Dans un souci de simplification, nous nous focaliserons ici sur les contrôleurs à une pièce.

Pour un joueur humain, une des difficultés de Tetris réside dans le fait que les pièces tombent relativement rapidement du haut de la zone de jeu : le peu de temps alloué rend parfois difficile la prise de décision. Cette dimension du problème n'apparaît pas quand il s'agit de construire un contrôleur informatique. Les contrôleurs que nous étudions dans nos expériences sont capables de jouer 50 000 à 100 000 coups par seconde sur une machine de bureau actuelle. La chute progressive des pièces est donc négligeable par rapport à la vitesse à laquelle un contrôleur est capable de prendre une décision. Comme dans la majorité des travaux sur Tetris (Tsitsiklis et van Roy, 1996; Bertsekas et Tsitsiklis, 1996; Kakade, 2001; Lagoudakis *et al.*, 2002; Ramon et Driessens, 2004; Farias et van Roy, 2006; Szita et Lőrincz, 2006), nous allons ignorer la chute des pièces et nous concentrer uniquement sur le cœur du problème, qui est de décider où placer chaque pièce qui se présente.

5.2 Principales approches

Dans la littérature, tous les travaux visant à concevoir des contrôleurs pour Tetris se fondent sur une **fonction d'évaluation**. Dans un **état** donné du jeu (c'est-à-dire la configuration actuelle du mur et la pièce courante), toutes les **actions** possibles (c'est-à-dire le choix d'une position et d'une orientation pour la pièce courante) sont évaluées à l'aide de cette fonction (voir figure 5.2). Le contrôleur choisit alors l'action qui a reçu la meilleure évaluation. L'évaluation est une note qui a pour but de discriminer les actions selon leur pertinence. On peut également créer un contrôleur à deux pièces à partir d'une fonction d'évaluation (voir figure 5.3).

Ainsi, le problème de créer un joueur de Tetris revient à concevoir une bonne fonction d'évaluation. Idéalement, on voudrait que cette fonction fournisse des valeurs élevées pour les bonnes décisions, et de faibles valeurs pour les mauvaises. L'évaluation est en général une combinaison de **fonctions de base**, typiquement une somme pondérée (mais pas toujours, voir par exemple (Böhm *et al.*, 2005)). Comme dans l'apprentissage par renforcement, les fonctions de base tentent de capturer les caractéristiques pertinentes

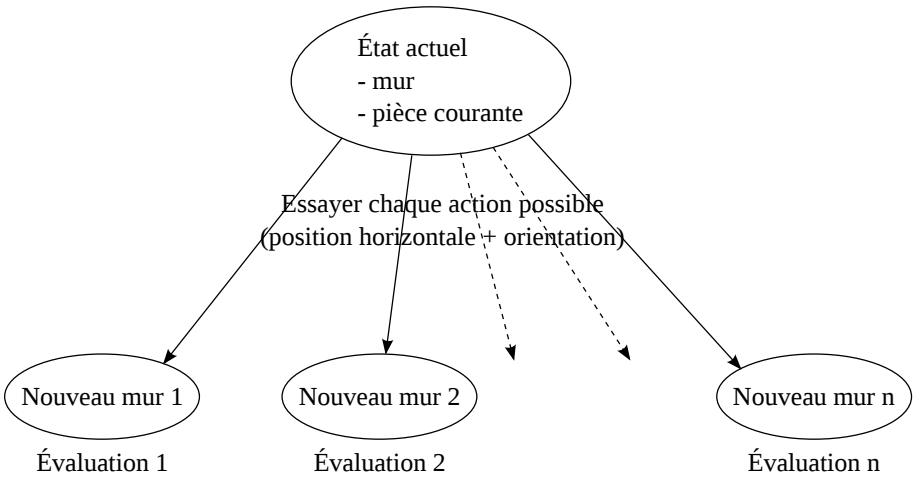


FIGURE 5.2 – Principe d'un contrôleur à une pièce. L'algorithme teste chaque action possible et l'évalue à l'aide d'une fonction d'évaluation, typiquement une combinaison linéaire de fonctions de base. L'action qui reçoit la meilleure évaluation est effectuée.

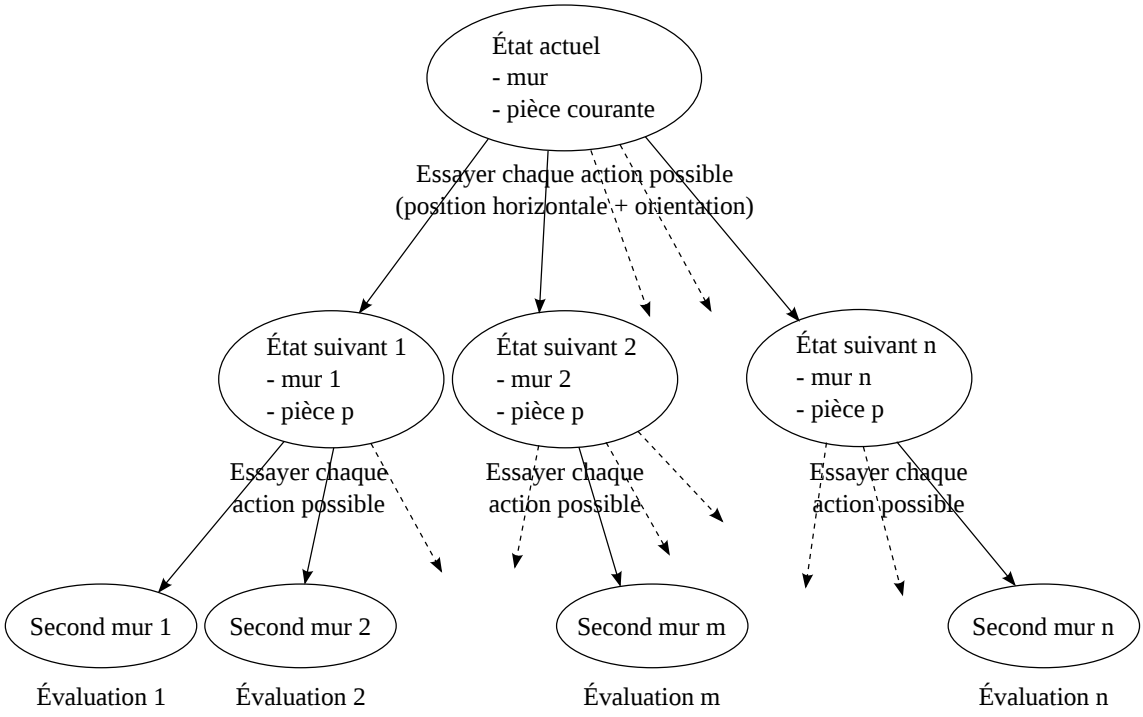


FIGURE 5.3 – Principe d'un contrôleur à deux pièces. L'algorithme connaît ici à l'avance la prochaine pièce. Pour chaque action possible, on connaît de façon déterministe l'état résultant (le mur et la nouvelle pièce). On teste alors à nouveau sur cet état toutes les actions possibles et on les évalue à l'aide de la fonction d'évaluation. La meilleure évaluation obtenue définit l'action qui est effectuée.

des actions et des états. Un exemple de fonction de base à Tetris est le nombre de trous [8] dans le mur résultant d'une décision. Son poids associé est habituellement négatif : plus il y a de trous, plus l'évaluation est basse car les trous empêchent de former des lignes.

Concevoir un contrôleur pour Tetris basé sur une fonction d'évaluation se fait généralement en deux étapes. La première étape consiste à choisir un ensemble de fonctions de base approprié, c'est-à-dire capable d'extraire les informations pertinentes du jeu, et elle est habituellement accomplie par un expert. Le tableau 5.1 fournit une liste des fonctions de base introduites et utilisées par les travaux dont nous avons connaissance. La seconde étape consiste à déterminer le poids de chaque fonction de base dans la somme pondérée. Dans la littérature, cette étape est faite manuellement (par un expert) ou automatiquement (par apprentissage par renforcement ou par des techniques d'optimisation). Nous présentons maintenant plus en détail les travaux les plus significatifs à notre connaissance.

5.2.1 Approches par apprentissage par renforcement

Il est assez immédiat de modéliser le problème de Tetris comme un Processus Décisionnel de Markov $< \mathcal{S}, \mathcal{A}, T, R, \gamma >$. Un état $s \in \mathcal{S}$ comporte la configuration complète du mur ainsi que la pièce courante parmi les sept pièces possibles. Une action $a \in \mathcal{A}$ est le choix de la position où l'on place la pièce courante, c'est-à-dire une orientation et une coordonnée horizontale. La récompense est le nombre de lignes réalisées. La fonction de transition $T(s, a, s')$ peut se décomposer en deux phases, l'une déterministe et l'autre non déterministe. La première phase, déterministe, définit le mur qui résulte de l'action a, et la seconde phase génère aléatoirement une nouvelle pièce avec une distribution uniforme parmi les pièces existantes. Le nouvel état s' est alors constitué par ce nouveau mur et cette nouvelle pièce. La fonction de récompense $R(s, a, s')$ est quant à elle déterministe puisque pour un état s donné, on peut déterminer à l'avance le nombre de lignes complétées par toute action a. Elle ne dépend pas de l'état s'. Il faut noter que les fonctions de transitions et de récompense T et R sont connues dans le cas du problème de Tetris. Enfin, nous avons vu que toute partie à Tetris se terminait nécessairement (Burgiel, 1997), donc on peut prendre un facteur d'actualisation $\gamma = 1$. A partir de cette modélisation, les résultats concernant les PDM s'appliquent au problème de Tetris : on sait qu'il existe une politique optimale à Tetris, et comme $\gamma = 1$, la fonction de valeur $V^{\pi}(s)$ d'un état représente le nombre de lignes moyen qui peuvent être réalisées jusqu'à la fin de la partie en suivant la politique π.

Plusieurs travaux de la littérature d'apprentissage par renforcement avec approximation de fonction considèrent le jeu de Tetris. Certains d'entre eux sont mentionnés dans une revue de la littérature (Carr, 2005). La fonction d'évaluation que l'on choisit pour sélectionner les actions (voir figure 5.2) est ici la fonction de valeur, c'est-à-dire l'espérance du score futur à partir de chaque état. Ces algorithmes visent ainsi à fixer les poids de manière à ce que la somme des fonctions de base approxime la fonction de valeur, comme nous l'avons vu dans les chapitres précédents. Comme le nombre d'états est très élevé, l'espace d'états est exploré à l'aide de simulations. La première approche concernant Tetris dans ce domaine semble être due à Tsitsiklis et van Roy (1996). Leurs travaux font appel à une version approximative de Value Iteration. Le contrôleur obtenu utilise deux fonctions de base (la hauteur maximale du mur et le nombre de trous) et réalise un faible score moyen (environ une trentaine de lignes). Cependant, leur travail a montré qu'utiliser une fonction de valeur paramétrée linéairement fonctionne mieux que de choisir simplement l'action qui maximise la récompense immédiate (rappelons que celle-ci est déterministe et connue) : selon eux, la plupart du temps, cette méthode ne parvient pas à faire une seule ligne.

Par la suite, Bertsekas et Ioffe (1996) ont proposé l'algorithme λPI (voir la section 1.2.4) et ont appliqué à Tetris sa version approximative AλPI telle que nous l'avons décrite à la section 3.4.3). La fonction de valeur est approchée par un ensemble de fonctions de base plus élaboré (voir tableau 5.1), et elle est estimée à l'aide de simulations. Leur approche permet d'atteindre un score moyen de 3 200 lignes sur 100 parties jouées. Cet ensemble de fonctions de base a été réutilisé par la suite dans d'autres travaux dont deux d'apprentissage par renforcement. Dans le premier, Kakade (2001) a appliqué la méthode Natural Policy Gradient et a mesuré un score moyen d'environ 6 800 lignes par partie, sans cependant spécifier sur combien de parties cette moyenne a été réalisée. Dans le second, Farias et van Roy (2006)

8. Un trou est une cellule vide recouverte par une cellule pleine.

8. Un puits est une succession de cases vides dans une colonne, telles que leurs deux cellules voisines à droite et à gauche sont occupées. Les puits profonds sont pénalisants car ils forcent à attendre une barre verticale ou à faire des trous.

Fonction de base	**Description**	Tsitsiklis et van Roy, 1996	Bertsekas et Tsitsiklis, 1996	Llima, 2005	Lagoudakis *et al.*, 2002	Fahey, 2003	Dellacherie (Fahey, 2003)	Ramon et Driessens, 2004	Böhm *et al.*, 2005	Thiery et Scherrer, 2009a
Hauteur maximale	Hauteur maximale d'une colonne	×	×	×	×			×	×	
Trous	Nombre de cellules vides surmontées d'une cellule pleine	×	×	×	×	×	×	×	×	×
Hauteur de colonne	Hauteur de chaque colonne		×					×		
Différence de colonne	Différence de hauteur entre chaque paire de colonnes adjacentes		×					×		
Hauteur d'arrivée	Hauteur à laquelle la dernière pièce a été posée			×			×		×	×
Transitions	Nombre de transitions plein-vide ou vide-plein entre deux cellules voisines			×						
Profondeur des puits	Somme des profondeurs des puits, sauf pour les puits de profondeur 1			×						
Trous pondérés	Sorte de somme pondérée des trous (manque de documentation)			×						
Somme des différences	Somme des différences de hauteur entre deux colonnes adjacentes				×					
Hauteur moyenne	Hauteur moyenne des colonnes				×			×		
Δ hauteur maximale	Variation de la hauteur maximale par rapport au coup suivant				×					
Δ trous	Variation du nombre de trous par rapport au coup suivant				×					
Δ somme des différences	Variation de la somme des différences par rapport au coup suivant				×					
Δ hauteur moyenne	Variation de la hauteur moyenne par rapport au coup suivant				×					
Lignes	Nombre de lignes supprimées au dernier coup				×	×			×	
Cellules pondérées	Cellules pleines pondérées par leur hauteur					×			×	
Puits	Somme de la profondeur des puits					×		×	×	
Cellules pleines	Nombre de cellules pleines					×			×	
Érosion	(Nombre de lignes supprimées au dernier coup) × (Nombre de cellules éliminées dans la dernière pièce ajoutée)						×			×
Transitions horizontales	Nombre de transitions entre cellules d'une même ligne						×		×	×
Transitions verticales	Nombre de transitions entre cellules d'une même colonne						×		×	×
Puits cumulatifs	$\sum_{p \in \text{puits}}(1 + 2 + \cdots + \text{profondeur}(p))$						×			×
Hauteur minimale	Hauteur minimale d'une colonne							×		
H. max − H. moyenne	Hauteur maximale − Hauteur moyenne							×		
H. moyenne − H. min	Hauteur moyenne − Hauteur minimale							×		
Prof. moyenne des trous	Profondeur moyenne des trous sous la surface							×		
Différence maximale	Différence maximale de hauteur entre deux colonnes								×	
Trous adjacents	Nombre de trous, en comptant comme un seul trou plusieurs trous adjacents dans la même colonne								×	
Puits le plus profond	Profondeur maximale d'un puits								×	
Profondeur des trous	Nombre de cellules pleines au-dessus de chaque trou									×
Lignes avec trous	Nombre de lignes avec au moins un trou									×
Diversité de motifs	Nombre de motifs de transitions différents entre deux colonnes adjacentes									×
Constante	Terme constant égal à 1	×	×		×			×		

TABLE 5.1 – Liste de fonctions de bases utilisés par les principaux travaux sur Tetris.

ont appliqué une approche de programmation linéaire, atteignant un score moyen de 4 700 lignes sur 90 parties jouées. Lagoudakis et Parr (2003) ont quant à eux appliqué l'algorithme LSPI présenté au chapitre 3 avec des fonctions de base originales et définies sur l'espace des couples états-actions. Leurs expériences aboutissent à « un score moyen de 1 000 à 3 000 lignes ». L'algorithme étant off-policy, il collecte des échantillons d'apprentissage une seule fois et ces échantillons ont l'avantage de pouvoir être réutilisés à chaque changement de politique. Même s'il n'a pas abouti à des performances élevées, nous pouvons également mentionner le travail de Ramon et Driessens (2004) basé sur de l'apprentissage par renforcement relationnel.

5.2.2 Approches d'optimisation générale

Une alternative à l'apprentissage par renforcement pour déterminer les poids est d'utiliser des techniques d'optimisation générale, où un algorithme cherche directement des poids tels que le contrôleur correspondant soit performant. Contrairement à l'apprentissage par renforcement où la fonction d'évaluation est une approximation du score futur, la fonction d'évaluation n'a ici pas nécessairement de sémantique. Du point de vue de l'apprentissage par renforcement, il s'agit d'une recherche directe dans l'espace des politiques. Par exemple, le programme de l'implémentation GNU Xtris utilise 6 fonctions de base dont les poids ont été optimisés grâce à des algorithmes génétiques (Llima, 2005). Cet algorithme a fait évoluer 50 ensembles de coefficients sur 18 générations, pendant 500 heures-machines distribuées sur 20 stations de travail. Sur un simulateur très proche du Tetris original, ce programme réalise une moyenne de 50 000 lignes par partie. Böhm *et al.* (2005) reportent également des résulats avec une approche évolutionnaire en utilisant des fonctions de base de la littérature ainsi que quelques fonctions de base originales. Cependant, leurs résultats ne peuvent pas être comparés avec la plupart des autres travaux car ils considèrent uniquement des contrôleurs à deux pièces. De plus, pour des raisons de temps d'exécution, ils ne fournissent aucun résultat précis sur la grille standard de taille 10×20. Enfin, Szita et Lőrincz (2006) ont appliqué la méthode d'entropie croisée (voir de Boer *et al.*, 2004), une méthode proche des algorithmes évolutionnaires, où une population de contrôleurs évolue autour d'une distribution gaussienne. Ils ont réutilisé les fonctions de base de Bertsekas et Ioffe (1996), et ont obtenu un score moyen de 350 000 lignes par partie (sur 30 parties), dépassant ainsi les approches par apprentissage par renforcement qui utilisent ces mêmes fonctions de base. Nous détaillons leur approche dans la section 5.4.

5.2.3 Contrôleurs réglés manuellement

À notre connaissance, le meilleur contrôleur à une pièce de l'état de l'art est celui de Dellacherie (Fahey, 2003) et a été paramétré à la main. Dellacherie a mis au point un ensemble efficace de fonctions de base et a fixé ses poids manuellement. Étonnamment, ce joueur réglé à la main dépasse les performances des travaux d'apprentissage par renforcement et les résultats de Szita et Lőrincz (2006) : sur un total de 56 parties, l'algorithme de Dellacherie a atteint un score moyen d'environ 660 000 lignes. De plus, il faut noter que cette mesure de 660 000 lignes par partie n'a pas été réalisée avec la version simplifiée usuelle de Tetris décrite plus haut, mais avec une implémentation du Tetris original qui, comme nous allons le voir dans la section 5.3, est plus difficile que le problème de Tetris simplifié considéré par la plupart des travaux de recherche. Le code source étant mis à disposition par Colin Fahey (Fahey, 2003), nous l'avons analysé pour déterminer les fonctions de base et leurs poids. La fonction d'évaluation de Dellacherie est la somme pondérée suivante :

$$\begin{aligned} &- (\text{Hauteur d'arrivée}) + (\text{Érosion}) - (\text{Transitions horizontales}) \\ &- (\text{Transitions verticales}) - 4 \times (\text{Nombre de trous}) - (\text{Puits cumulatifs}) \end{aligned}$$

où les fonctions de base ci-dessus sont détaillées dans le tableau 5.1.

Cet aperçu de l'état de l'art donne lieu à quelques commentaires. D'une part, les fonctions de base de Dellacherie (Fahey, 2003) semblent être les plus efficaces car même avec des poids fixés à la main, le contrôleur de Dellacherie a jusqu'ici donné les meilleurs résultats. D'autre part, les algorithmes d'optimisation (Szita et Lőrincz, 2006; Böhm *et al.*, 2005; Llima, 2005) apparaissent comme les méthodes les plus performantes pour fixer les poids d'un ensemble d'un ensemble de fonctions de base donné. La

raison pour laquelle les approches d'apprentissage par renforcement (qui cherchent à exploiter la structure du problème de Tetris) n'atteignent pas d'aussi bonnes performances est probablement que, pour les techniques de l'état de l'art, le problème de Tetris est encore trop difficile à traiter. Il se peut que la fonction de valeur soit trop difficile à estimer avec une architecture linéaire. Or, rechercher la fonction de valeur optimale n'est qu'un moyen indirect d'optimiser π : dans l'exemple de Tetris, il semble qu'explorer directement l'espace des politiques soit plus efficace.

5.3 Difficulté de comparer les joueurs artificiels

Nous venons de mentionner les scores moyens annoncés par les auteurs de plusieurs contrôleurs de Tetris. Dans cette section, nous mettons en évidence le fait que comparer des contrôleurs de Tetris variés est un problème délicat, en particulier lorsque leurs performances sont mesurées sur des implémentations différentes. D'abord, nous soulignons le fait que les spécifications du jeu diffèrent souvent entre les travaux. Ensuite, nous montrons que le score moyen d'un contrôleur de Tetris a une grande variance et nous expliquons comment obtenir des intervalles de confiance. Enfin, nous remarquons que la performance d'un contrôleur peut varier significativement à cause de certains détails subtils dans l'implémentation d'un contrôleur.

5.3.1 Spécification du jeu

Nous avons déjà mentionné le fait que la plupart des travaux mettent au point des contrôleurs à une pièce et que certains considèrent des contrôleurs à deux pièces. Fahey (2003), qui propose un contrôleur à deux pièces, explique que lorsque la prochaine pièce n'est plus connue, le contrôleur à une pièce correspondant réalise des mauvais scores comparés aux autres contrôleurs à une pièce de la littérature. Cela suggère que la connaissance de la prochaine pièce augmente considérablement la performance. Cela signifie également que les travaux qui utilisent des contrôleurs à une pièce sont sous-évalués si l'on compare leurs résultats aux quelques travaux qui se basent sur des contrôleurs à deux pièces (Fahey, 2003; Böhm *et al.*, 2005) [9].

Lorsque l'on s'intéresse au problème de Tetris tel qu'il est considéré par la plupart des chercheurs, on peut remarquer qu'il est légèrement différent par rapport au jeu de Tetris original spécifié par Fahey (2003). Certaines simplifications usuelles sont effectuées afin de se focaliser sur la préoccupation essentielle d'un joueur artificiel, qui est de choisir la position et l'orientation de la pièce courante. Dans le jeu original de Tetris (voir Fahey, 2003), la pièce courante apparaît à l'intérieur de la zone de jeu, puis descend graduellement. La partie est perdue lorsque la pièce n'a pas assez d'espace pour apparaître en haut de la grille. Comme dans la majorité des travaux (Tsitsiklis et van Roy, 1996; Bertsekas et Tsitsiklis, 1996; Kakade, 2001; Lagoudakis *et al.*, 2002; Ramon et Driessens, 2004; Farias et van Roy, 2006; Szita et Lőrincz, 2006), nous considérons une simplification de Tetris : le contrôleur décide simplement dans quelle colonne et avec quelle orientation il lâche la pièce. De cette manière, le jeu est légèrement simplifié car la pièce apparaît dans la zone de jeu directement là où le joueur a décidé de la placer. Ainsi, on considère qu'il y a toujours de l'espace au-dessus de la zone de jeu pour choisir l'orientation et la colonne où l'on va lâcher la pièce. Cela produit une différence importante, car l'intégralité de l'espace de la grille devient utilisable, y compris les lignes les plus hautes. On évite ainsi les situations où la hauteur du mur empêcherait la pièce d'atteindre un côté de la grille. Ce jeu de Tetris simplifié est donc plus facile que le jeu original, et un contrôleur a la possibilité de réaliser un plus grand nombre de lignes [10].

5.3.2 Grande variance des scores à Tetris

Bien que la plupart des auteurs semblent conscients du fait que les scores à Tetris ont une importante variance, presque aucun ne fournit des intervalles de confiance. A notre connaissance, le travail de Szita

9. Rappelons qu'un contrôleur à une pièce peut facilement être étendu à un contrôleur à deux pièces (voir figure 5.3)

10. Notons qu'avec cette simplification usuelle, il devient impossible de reboucher un « trou » par le côté en laissant la pièce courante descendre et en la déplaçant ensuite horizontalement. Cependant, cela n'a pas de conséquence dans notre étude car les contrôleurs implémentés sur le simulateur de Fahey (2003) du jeu de Tetris original ne tirent pas non plus parti de cette possibilité.

et Lőrincz (2006) est le seul à le faire. Nous expliquons ici comment calculer ces intervalles de confiance.

Fahey (2003) a conjecturé que le score d'une partie de Tetris pour un contrôleur fixé suit une distribution exponentielle. Le score (le nombre de lignes) étant un nombre entier, une conjecture plus juste serait de dire qu'il suit une distribution géométrique. Intuitivement, dans les deux cas, l'idée est de considérer que la hauteur du mur au cours d'une partie suit une sorte de marche aléatoire : elle augmente et diminue selon les pièces qui tombent. La fin de la partie, et donc le score final, sont déterminés par le moment où cette marche aléatoire atteint le haut de la zone de jeu. Il est établi que, dans une marche aléatoire, le moment où un point précis est atteint suit asymptotiquement une loi géométrique. Cela peut être vu comme une conséquence du théorème de Perron-Frobenius (Billingsley, 1995). En effet, lorsque le temps t tend vers l'infini, la probabilité que la marche aléatoire soit toujours dans un état non absorbant au temps t est égale à $a|\Lambda|^t$ où a est une constante et Λ est la valeur propre non unitaire de plus grand module de la matrice stochastique associée à la marche aléatoire. Par conséquent, la probabilité d'atteindre la fin de la marche aléatoire exactement au temps t est égale à $a|\Lambda|^{t+1} - a|\Lambda|^t$, qui est proportionnel à $|\Lambda|^t$.

Même si une loi géométrique serait plus appropriée qu'une loi exponentielle, la conjecture de Fahey a été validée expérimentalement par Szita et Lőrincz (2006) pour de nombreux contrôleurs (chaque contrôleur génère une distribution exponentielle des scores à 95 %) en utilisant le test statistique de Kolmogorov-Smirnov. Si l'on suppose une distribution exponentielle, l'écart-type est égal à l'espérance, et si l'on suppose une distribution géométrique, il est très proche de l'espérance (voir par exemple Billingsley, 1995). En effet, une loi géométrique de paramètre p a pour espérance $\frac{1}{p}$ et pour écart-type $\sqrt{\frac{1-p}{p^2}}$. Si p est très petit, ce qui est le cas pour des contrôleurs de Tetris performants, l'écart-type est très proche de $\frac{1}{p}$.

Le fait que l'écart-type soit très proche de l'espérance nous permet de déduire un intervalle de confiance. Lorsque l'on évalue le score moyen d'un contrôleur en jouant un certain nombre de parties, la confiance que l'on peut accorder à l'estimation augmente avec le nombre de parties mais diminue si on augmente l'écart-type. Comme l'écart-type est très proche de l'espérance, plus un contrôleur est performant, plus il est difficile d'évaluer précisément ses performances.

Plus précisément, l'intervalle de confiance a la forme suivante. Avec probabilité p, la différence entre la moyenne empirique $\hat{\mu}$ de N parties jouées et l'espérance réelle μ satisfait

$$|\mu - \hat{\mu}| \leq \frac{k\sigma}{\sqrt{N}} = \frac{k\mu}{\sqrt{N}} \simeq \frac{k\hat{\mu}}{\sqrt{N}}$$

où σ est l'écart-type du score, et k est une constante qui dépend de la probabilité p (typiquement, $k = 1$ pour $p = 0.68$, $k = 2$ pour $p = 0.95$, $k = 3$ pour $p = 0.997$). Cela nous conduit à l'intervalle de confiance relatif suivant :

$$\frac{|\mu - \hat{\mu}|}{\hat{\mu}} \leq \frac{k}{\sqrt{N}}.$$

Illustrons cet intervalle de confiance en considérant l'évaluation du contrôleur de Dellacherie sur $N = 56$ parties. On peut conclure de l'analyse ci-dessus que l'intervalle de confiance de la moyenne empirique (660 000 lignes) est de ± 27 % avec probabilité $0,95$. Malgré sa grande taille, l'intervalle de confiance confirme qu'avec grande probabilité $(0,95)$, le joueur artificiel de Dellacherie reste le meilleur.

Plus généralement, les contrôleurs évalués avec $N = 100$ parties donnent un intervalle de confiance de $\pm 20\%$ valide 95% du temps. Dans le reste de cette étude, nous choisissons d'utiliser la notation $m \pm c\%$ pour représenter des intervalles de confiance valides 95% du temps (c'est-à-dire $k = 2$), afin de fournir des intervalles intuitifs pour des scores de Tetris, valables la plupart du temps.

5.3.3 Subtilités d'implémentation

Nous venons de voir qu'en général, les intervalles de confiance que l'on peut calculer pour des joueurs artificiels pour Tetris sont assez grands. Lorsque nous avons implémenté notre propre simulateur de Tetris, nous avons également remarqué que certains détails subtils dans l'implémentation pouvaient avoir un effet significatif sur les résultats obtenus.

Un premier détail subtil (qui n'est jamais explicité dans les travaux dont nous avons connaissance) est de savoir comment un joueur artificiel utilisant une fonction d'évaluation se comporte lorsqu'il est proche

de perdre la partie. Il se peut qu'à un moment donné, la décision qui a reçu la meilleure évaluation fasse perdre la partie immédiatement, alors que d'autres décisions (avec des évaluations plus basses) auraient permis de continuer le jeu plus longtemps. Dans un tel cas, il est préférable de ne pas considérer comme des actions possibles celles qui mènent directement à la fin de la partie : la partie dure alors plus longtemps et le score sera meilleur. Si l'on choisit de procéder de cette manière, cela signifie que la partie est perdue si et seulement si toutes les décisions font perdre la partie. C'est le choix que nous avons fait. Sur notre simulateur, le joueur artificiel de Dellacherie réalise ainsi 5 200 000 ± 20 % lignes en moyenne. Si nous laissons le joueur artificiel effectuer des actions qui font perdre (en ayant la meilleure évaluation), les résultats de Dellacherie tombent à 850 000 ± 20 % lignes.

De plus, la plupart des implémentations de Tetris définissent la fin de la partie comme étant le moment où la pièce courante n'a pas assez d'espace pour être placée dans la zone de jeu, c'est-à-dire lorsque la pièce déborde de la grille de taille 10×20. C'est la définition que nous avons adoptée. Cependant, si nous examinons de plus près la description de Tetris écrite par Bertsekas et Ioffe (1996), nous pouvons voir qu'ils considèrent que la partie *"se termine lorsqu'une case de la ligne du haut devient pleine et que le haut du mur atteint le haut de la grille"*. Cette définition est équivalente à dire que la pièce déborde d'une grille de taille 10×19. Ce genre de détail peut produire une différence significative sur les scores : avec une grille de taille 10×19, l'algorithme de Dellacherie réalise 2 500 000 ± 20 % lignes avec notre implémentation au lieu de 5 200 000 ± 20 %. Par conséquent, nous pensons que les résultats expérimentaux de Bertsekas et Ioffe (1996) avec AλPI, mentionnés dans la section 5.2, sont sous-estimés par rapport aux autres travaux d'apprentissage par renforcement.

Comme des petits détails concernant les règles du jeu et l'implémentation du joueur artificiel peuvent avoir des effets significatifs sur les scores, il est nécessaire d'employer le plus grand soin lorsque l'on compare différents joueurs artificiels. Le seul moyen d'effectuer une comparaison fiable de plusieurs contrôleurs est de les lancer sur le même simulateur et un grand nombre de fois. Pour ce faire, nous avons implémenté un simulateur de Tetris configurable et optimisé, ainsi que plusieurs contrôleurs [11].

Dans la suite de ce mémoire, nous comparons les résultats des contrôleurs considérés avec les performances du contrôleur de Dellacherie sur les deux configurations de jeu suivantes.

- La première configuration est le jeu de Tetris standard utilisé par la plupart des travaux de recherche. Nous utilisons une grille de taille 10×20 en considérant comme référence le meilleur score que nous connaissons : 5 200 000 ± 20 % lignes en moyenne, qui est le score atteint par le contrôleur de Dellacherie sur notre implémentation.
- La seconde configuration nous permet de déduire une borne inférieure sur le score de nos contrôleurs s'ils étaient lancés sur une implémentation du Tetris original tel que spécifié par Fahey (2003). Dans le Tetris original, les pièces doivent être déplacées étape par étape jusqu'à leur position finale. Comme expliqué dans la section 5.1, par rapport au Tetris simplifié, la principale difficulté est qu'il peut y avoir des problèmes de collisions lorsque le mur est trop haut. Comme la hauteur d'une pièce est toujours inférieure ou égale à 4 (voir figure 5.1), il est clair qu'un contrôleur capable d'atteindre un certain score sur notre implémentation avec une grille de taille 10×16 fera un meilleur score sur le Tetris original en taille 10×20. Pour cette raison, nous considérerons aussi les performances de nos contrôleurs sur un jeu de taille 10×16 et nous comparerons ces résultats avec le score de 660 000 ± 27 % lignes (obtenu par le contrôleur de Dellacherie sur le simulateur du Tetris original de Fahey (2003)). On note cependant que cette borne inférieure est assez pessimiste car, en taille 10×16, notre implémentation du contrôleur de Dellacherie ne réalise que 220 000 ± 20 % lignes.

5.4 La méthode d'entropie croisée

Malgré les réserves que nous venons d'émettre à propos de la comparaison de joueurs artificiels sur différentes implémentations, l'entropie croisée semble être actuellement l'approche la plus efficace pour régler les poids de la fonction d'évaluation d'un contrôleur pour Tetris. Szita et Lőrincz (2006) ont montré que cette méthode améliore les scores des travaux d'apprentissage par renforcement de plusieurs ordres de grandeur. Dans cette section, nous décrivons la méthode d'entropie croisée et nous expliquons comment Szita et Lőrincz (2006) l'ont appliquée à Tetris.

11. Le code source C est disponible ici : `http://gforge.inria.fr/projects/mdptetris`.

La description qui suit est inspirée de celle de Szita et Lőrincz (2006). Une description plus détaillée de cet algorithme d'optimisation peut être trouvée dans (de Boer *et al.*, 2004). La méthode d'entropie croisée est un algorithme stochastique itératif qui cherche à résoudre un problème d'optimisation de la forme :

$$w^* = \arg\max_w S(w)$$

où S est une fonction que l'on souhaite maximiser et w est un paramètre à optimiser à valeurs dans un espace continu (typiquement un vecteur).

La méthode d'entropie croisée itère sur une distribution de solutions et non sur une solution seule. On considère une famille de distributions $\mathcal{F}$ (par exemple les distributions gaussiennes) et on veut déterminer une distribution de probabilité $f \in \mathcal{F}$ qui génère des solutions w proches de la solution optimale w^*. A chaque itération t, on a une distribution $f_t \in \mathcal{F}$, et on veut que la distribution suivante $f_{t+1} \in \mathcal{F}$ produise de meilleures solutions. Pour cela, on considère qu'une solution w est une bonne solution si elle donne une valeur supérieure à un certain seuil γ_t, c'est-à-dire si $S(w) > \gamma_t$. Considérons alors g_{γ_t}, la distribution de probabilité uniforme qui génère des solutions dont les valeurs sont supérieures à γ_t. En général, cette distribution g_{γ_t} n'appartient pas à $\mathcal{F}$. L'idée de la méthode d'entropie croisée est de chercher la distribution $f_{t+1} \in \mathcal{F}$ qui s'en rapproche le plus, au sens de la *mesure d'entropie croisée*[12] (de Boer *et al.*, 2004). Pour de nombreux types de familles de distributions $\mathcal{F}$, cette distribution f_{t+1} peut être estimée à partir d'échantillons générés par la distribution f_t. Par exemple, dans le cas où $\mathcal{F}$ est l'ensemble des distributions gaussiennes, la distribution gaussienne la plus proche de g_{γ_t} est celle qui est caractérisée par la moyenne et la variance de la distribution g_{γ_t}. On peut estimer ces deux paramètres en générant des échantillons avec la distribution f_t et en sélectionnant ceux qui sont au-dessus du seuil γ_t, c'est-à-dire les meilleurs.

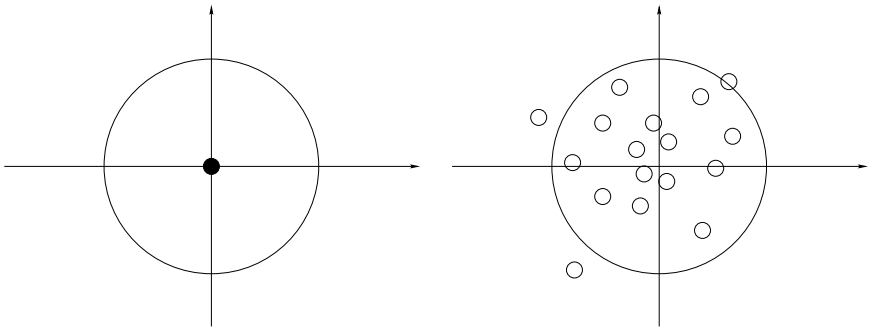
En pratique, la séquence des seuils γ_t est construite automatiquement en même temps que celle des distributions f_t. Précisément, la méthode d'entropie croisée dans le cas des distributions gaussiennes est détaillée dans l'algorithme 21, et une représentation graphique est donnée à la figure 5.4. Globalement, elle consiste à répéter les étapes suivantes. On génère N échantillons à partir de la distribution gaussienne actuelle f_t, et on évalue chacun de ces N vecteurs vis-à-vis de la fonction à optimiser S. On sélectionne ensuite une proportion $\rho \in]0, 1[$ des meilleures solutions (cela revient à fixer γ_t à un certain seuil). Puis on fixe les paramètres de la nouvelle distribution gaussienne f_{t+1} comme étant la moyenne et la variance empiriques des meilleures solutions sélectionnées. Un terme de **bruit** Z_t peut être ajouté à la mise à jour de la variance. Lorsque $Z_t \neq 0$, l'algorithme est appelé algorithme d'entropie croisée bruitée (de Boer *et al.*, 2004). On peut voir ce terme de bruit comme un moyen d'éviter une convergence trop rapide vers un mauvais optimum local.

Algorithme 21 Méthode d'entropie croisée bruitée avec une distribution gaussienne

evaluer() : une fonction qui estime la fonction à optimiser S pour un certain vecteur w
(μ, σ) : la moyenne et la variance de la distribution initiale
N : le nombre de vecteurs générés à chaque itération
ρ : la fraction de vecteurs sélectionnés
Z_t : le bruit ajouté à chaque itération

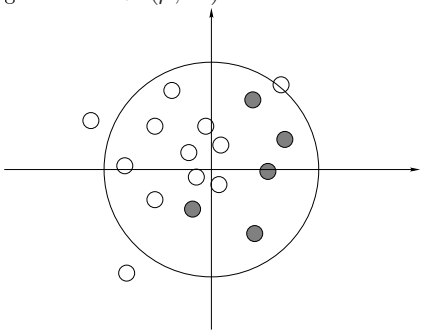
répéter
  Générer N vecteurs $w_1, w_2, \ldots, w_N$ selon $\mathcal{N}(\mu, \sigma^2)$
  Evaluer chaque vecteur à l'aide de la fonction evaluer()
  Sélectionner les $\lfloor \rho \times N \rfloor$ vecteurs ayant reçu les meilleures évaluations
  $\mu \leftarrow$ (moyenne des vecteurs sélectionnés)
  $\sigma^2 \leftarrow$ (variance des vecteurs sélectionnés) $+ Z_t$
fin répéter

12. La mesure d'entropie croisée (ou *divergence de Kullback-Leibler*) définit une notion de dissimilarité entre deux distributions de probabilités.

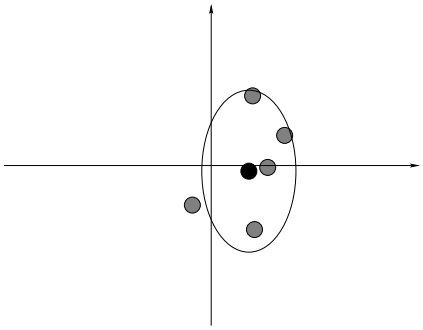


1. Commencer avec une distribution gaussienne $\mathcal{N}(\mu, \sigma^2)$.

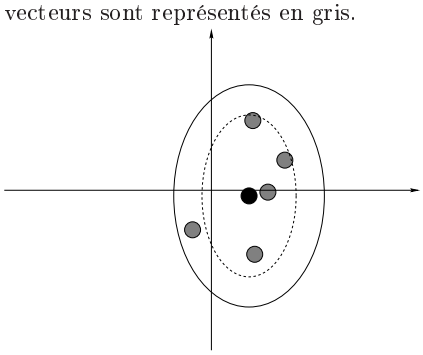
2. Générer N vecteurs avec cette distribution.



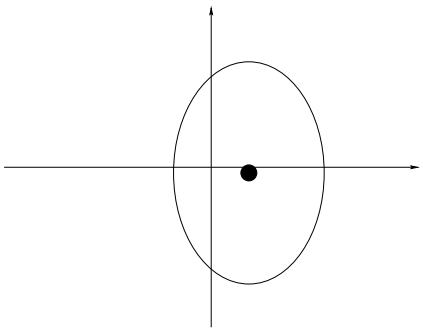
3. Evaluer chaque vecteur avec *evaluer*() et sélectionner une proportion ρ des meilleurs vecteurs. Ces vecteurs sont représentés en gris.



4. Calculer la moyenne et la variance des meilleurs vecteurs.



5. Ajouter un terme de bruit à la variance, afin d'éviter une convergence trop rapide vers un optimum local.



6. Cette moyenne et cette variance caractérisent une nouvelle distribution gaussienne qui générera des vecteurs pour la prochaine itération.

FIGURE 5.4 – Une représentation graphique de la méthode d'entropie croisée bruitée pour optimiser un vecteur à deux dimensions. La distribution gaussienne est representée avec un disque noir pour la moyenne et une ellipse pour la variance.

Dans la description de l'algorithme 21, la fonction evaluer() qui est utilisée pour évaluer chaque vecteur peut être S, ou bien une approximation de S si S prend trop de temps pour être calculée de manière exacte.

On peut remarquer que l'algorithme d'entropie croisée est relativement proche des algorithmes évolutionnaires. C'est en effet un processus itératif qui traite un ensemble de solutions candidates (ou individus). A chaque itération, les meilleurs individus sont sélectionnés, puis de nouveaux individus sont générés à partir d'eux. La principale particularité de la méthode d'entropie croisée est la manière dont les nouvelles solutions sont générées (selon une distribution, par exemple gaussienne).

Notons enfin que la méthode d'entropie croisée dans le cas gaussien est très proche de l'algorithme CMA-ES (Hansen et Ostermeier, 2001) (Covariance Matrix Adaptation Evolution Strategy). La différence essentielle réside dans le processus de mise à jour de la distribution gaussienne. La méthode d'entropie croisée se base uniquement sur les individus à l'itération courante pour mettre à jour la variance, et y ajoute un terme de bruit pour éviter de contracter vers un optimum local. CMA-ES cherche également à échapper à un processus contractant, mais en exploitant la forme des étapes précédentes pour générer une nouvelle étape espérée performante. De plus, l'algorithme CMA-ES permet de prendre en compte une distribution gaussienne avec une matrice de covariance complète, alors que la distribution gaussienne utilisée dans la méthode d'entropie croisée correspond au cas d'une matrice de covariance diagonale. Sur la figure 5.4 (page 89), les axes des ellipses représentées sont toujours parallèles à un axe du repère. Avec l'algorithme CMA-ES, les axes des ellipses peuvent être dans n'importe quelle direction.

Application à Tetris

Szita et Lőrincz (2006) ont appliqué la méthode d'entropie croisée bruitée avec une distribution gaussienne au problème de Tetris. Ils considèrent un contrôleur qui utilise les 22 fonctions de base de Bertsekas et Ioffe (1996), car ce type de contrôleur avait déjà été utilisé dans plusieurs travaux (Bertsekas et Tsitsiklis, 1996; Kakade, 2001; Farias et van Roy, 2006). Notons $w = (w_1, \ldots, w_{21})$ le vecteur de poids à déterminer. Pour cette application, la fonction $w \mapsto S(w)$ à optimiser est l'espérance du score atteint par le contrôleur correspondant au vecteur de poids w. Szita et Lőrincz (2006) partent d'une distribution gaussienne centrée à $\mu = (0, 0, \ldots, 0)$ avec variance $\sigma^2 = (100, 100, \ldots, 100)$. A chaque itération, ils génèrent $N = 100$ vecteurs et évaluent chacun d'entre eux en jouant une partie. Ils sélectionnent les 10 meilleurs vecteurs ($\rho = 10$ %) et génèrent ainsi la nouvelle distribution gaussienne. Après chaque itération, ils jouent 30 parties avec le vecteur de poids moyens de la nouvelle distribution, afin d'obtenir une courbe d'apprentissage représentant l'évolution des performances au fur et à mesure des itérations.

Dans l'expérience de Szita et Lőrincz (2006), la fonction evaluer() est le score d'*une seule* partie. Szita et Lőrincz (2006) ont lancé la méthode d'entropie croisée dans trois conditions expérimentales : sans bruit ($Z_t = 0$), avec un bruit constant ($Z_t = 4$), et avec un bruit linéairement décroissant ($Z_t = \max(5 - t/10, 0)$). Les formules des bruits constant et linéairement décroissant ont été fixées à la suite d'expériences préliminaires. Leurs résultats indiquent que les performances sont significativement améliorées lorsque l'on utilise du bruit. Leur meilleur contrôleur a été obtenu avec le bruit linéairement décroissant, atteignant un score moyen de $350\ 000 \pm 37$ % lignes. Nous détaillerons dans le chapitre 6 comment nous avons approfondi ces expériences pour améliorer encore les performances de la méthode d'entropie croisée sur Tetris.

Conclusion

Dans ce chapitre, nous avons présenté ce qui est à notre connaissance la première revue détaillée de la littérature concernant le problème de créer un contrôleur pour le jeu de Tetris. Nous avons résumé les travaux les plus significatifs dans divers domaines (l'apprentissage par renforcement, les algorithmes d'optimisation et les contrôleurs construits à la main) et dressé une liste complète des fonctions de base utilisées par ces travaux. De plus, nous avons mis en évidence la difficulté de comparer des résultats à Tetris. Évaluer des contrôleurs demande le plus grand soin en raison de la variance importante des scores réalisés, et du fait que des petits détails dans l'implémentation des règles du jeu peuvent avoir un effet

significatif sur les performances. Cet état de l'art peut servir de base pour les lecteurs intéressés par le problème de Tetris.

Une conclusion de cette revue de la littérature est que la connaissance experte que l'on apporte à un contrôleur (les fonctions de base) est au moins aussi déterminante sur les performances réalisées que le choix des paramètres de l'architecture (les coefficients appliqués aux fonctions de base). En effet, parmi les différentes approches connues, le contrôleur de Dellacherie (Fahey, 2003) atteint jusqu'ici les meilleurs scores avec des coefficients fixés manuellement.

Chapitre 6

Nouveaux résultats sur Tetris

Nous avons dressé dans le chapitre 5 un état de l'art du problème de Tetris, de ses particularités et des principaux travaux qui s'y sont intéressés. Nous présentons maintenant des résultats originaux que nous avons obtenus avec différentes approches. D'abord, nous considérons une taille de jeu réduite où le contrôle optimal stochastique exact devient applicable. Ensuite, nous revenons au jeu en taille normale sur lequel nous appliquons l'algorithme LSλPI introduit dans le chapitre 4. Nos résultats mettent en évidence l'intérêt de notre contribution par rapport à LSPI (Lagoudakis et Parr, 2003). Enfin, nous revisitons également la méthode d'entropie croisée appliquée à Tetris (Szita et Lőrincz, 2006) et nous expliquons comment, en la combinant avec des fonctions de base pertinentes, nous avons mis au point un contrôleur qui dépasse les performances des travaux précédents et qui nous a permis de remporter l'épreuve de Tetris de la compétition d'apprentissage par renforcement (Reinforcement Learning Competition) en 2008.

6.1 Contrôle optimal exact

Avant de considérer la taille normale du jeu (10×20) sur laquelle il est nécessaire d'employer des algorithmes d'apprentissage par renforcement approché, on peut s'intéresser à une taille réduite du jeu et calculer exactement la fonction de valeur optimale de ce problème réduit. Nous avons considéré un jeu de taille 5×5. Le nombre d'états est alors d'un peu moins de 7×2^{25}, soit environ $2,3 \times 10^8$. Dans ces conditions, il est alors possible de calculer une politique optimale de façon exacte en utilisant les algorithmes traditionnels du contrôle optimal stochastique présentés au chapitre 1. Nous avons ainsi exécuté Value Iteration sur ce problème réduit et, après une centaine d'heures de calculs sur une machine de bureau conventionnelle[13], nous avons obtenu une politique optimale. En taille 5×5, nous pouvons ainsi dire que la valeur de l'état initial (c'est-à-dire la zone de jeu entièrement vide), autrement dit le score moyen de la politique optimale, est d'environ $13,70$ lignes[14]. La convergence de la fonction de valeur n'étant qu'asymptotique, l'algorithme a été arrêté lorsque la distance $\|V_k - V_{k-1}\|_\infty$ entre deux fonctions de valeur successives est devenue inférieure au seuil $\epsilon = 10^{-6}$.

6.2 Approximation linéaire : LSλPI

Revenons à la taille de jeu normale (10×20) et intéressons-nous maintenant au contrôle optimal approché et à l'algorithme LSλPI, notre contribution présentée au chapitre 4. LSPI (le cas $\lambda = 1$) a été appliqué au jeu de Tetris par Lagoudakis *et al.* (2002). Nos résultats vont vérifier que diminuer λ permet de réduire la variance des estimations, et donc d'améliorer les performances lorsque le nombre d'échantillons est faible.

13. Processeur : Intel Core 2 Duo à $2,4$ GHz ; Mémoire vive : $3,4$ Go

14. L'animation qui se trouve en bas à droite de chaque page impaire de ce manuscrit représente une partie jouée par le contrôleur optimal ainsi obtenu sur le jeu de taille 5×5. Au cours de cette partie, 43 lignes ont été réalisées.

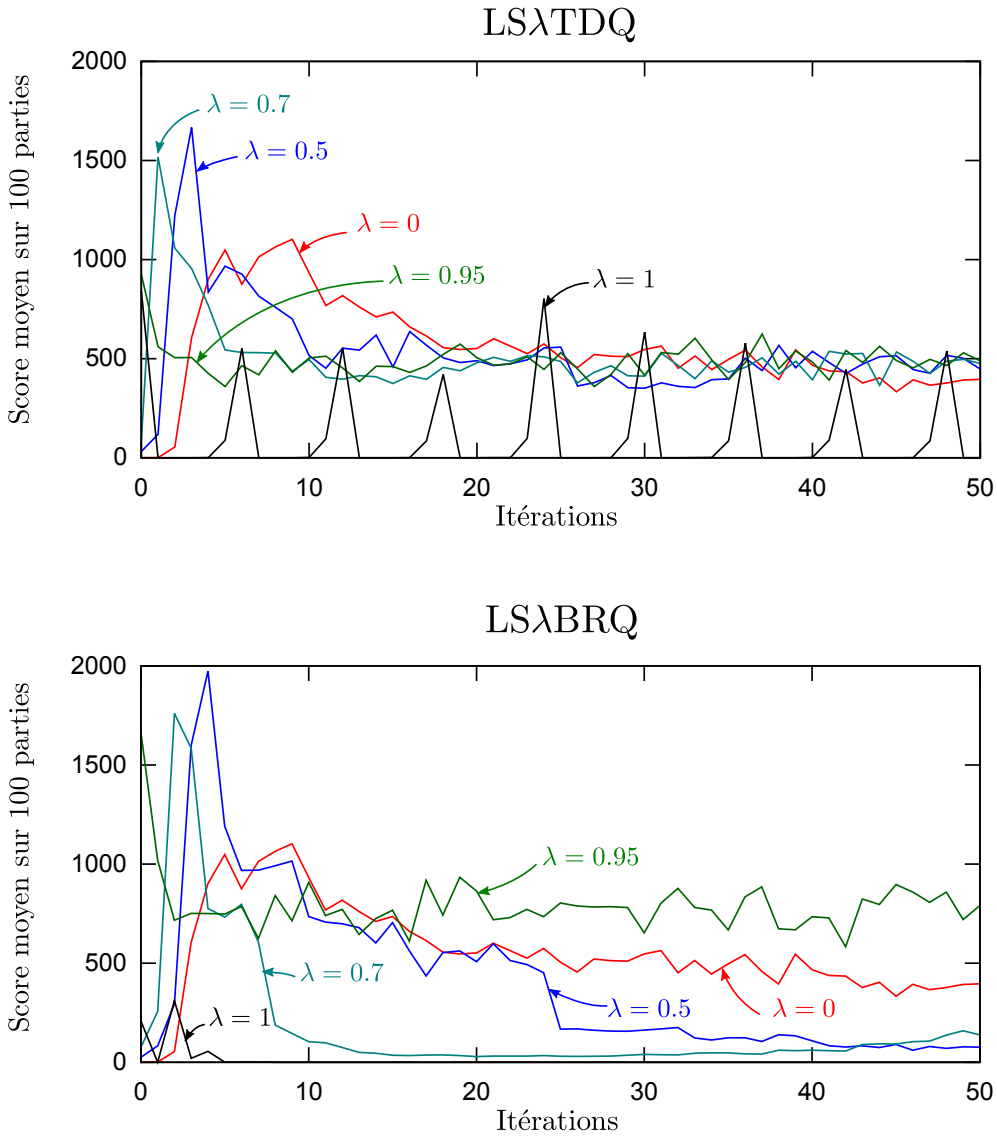


FIGURE 6.1 – Score moyen de 100 parties de Tetris pour différentes valeurs de λ à chaque itération de LSλPI. A cause du faible nombre d'échantillons (1000), l'algorithme diverge lorsque $\lambda = 1$ pour les deux méthodes. C'est avec LSλBRQ que la meilleure performance est atteinte (800 lignes de moyenne), pour $\lambda = 0,9$. Certaines courbes présentent des pics dans les premières itérations et baissent par la suite. On peut penser que la politique, devenue performante, est également devenue plus difficile est évaluer.

Nous avons reproduit le protocole expérimental de Lagoudakis *et al.* (2002). Nous avons ainsi lancé des expériences avec les mêmes fonctions de base et en utilisant la connaissance du modèle du PDM. Les fonctions de base, définies sur l'espace des couples états-actions, sont (voir aussi le tableau 5.1 page 83) :

- la hauteur maximale de la pile,
- le nombre de trous,
- la somme des différences de hauteur entre colonnes adjacentes (en valeur absolue),
- la hauteur moyenne des colonnes,
- le changement de ces quantités dans l'état suivant (afin de capturer l'effet du choix d'une action depuis l'état courant),
- le nombre de lignes réalisées en effectuant l'action,
- un terme constant.

Bien que notre politique initiale soit la même que celle de Lagoudakis *et al.* (2002) (communication personnelle), les scores peuvent difficilement être comparés. La politique initiale réalise environ 250 lignes de moyenne par partie sur notre implémentation, tandis qu'ils reportent un score initial moyen de 600 lignes. Ceci est vraisemblablement dû à des différences d'implémentation qui peuvent avoir un impact significatif sur le score (voir section 5.3.3).

Nous avons d'abord lancé LSλPI sur un ensemble de 10 000 échantillons, comme Lagoudakis *et al.* (2002) l'ont fait pour LSPI (c'est-à-dire $\lambda = 1$). Nous avons observé que diminuer λ n'améliorait pas la performance (cela ne faisait que ralentir la convergence). On peut supposer que l'ensemble d'échantillons était suffisamment grand pour éviter les problèmes de variance. Réduire λ n'est pas utile dans ce cas. Nous avons donc ensuite employé un ensemble d'échantillons plus réduit (1 000 échantillons au lieu de 10 000) afin de rendre la convergence plus difficile. La figure 6.1 représente la performance des politiques apprises pour différentes valeurs de λ. Lorsque $\lambda = 1$, l'algorithme est très instable et génère de mauvaises politiques car le nombre d'échantillons est faible, ce qui rend la variance de l'estimation importante. Le score oscille entre 0 et 600 lignes par partie avec LSλTDQ, et tombe à 0 avec LSλBRQ. De meilleures performances sont atteintes pour d'autres valeurs de λ. Comme pour le problème de la chaîne d'états (section 4.2), on remarque que λ a plus d'influence dans le cas de LSλBRQ. Après convergence, le meilleur score moyen est atteint avec $\lambda = 0,9$ et en utilisant LSλBRQ. La politique correspondante réalise environ 800 lignes par partie (rappelons que la politique initiale atteignait environ 250 lignes par partie).

Nos résultats expérimentaux confirment donc l'influence de λ sur la qualité de l'approximation et la performance des politiques générées. Les valeurs de λ intermédiaires (différentes de 0 et 1) peuvent en effet donner de meilleurs résultats en pratique lorsque le nombre d'échantillons est limité.

On remarque que par ailleurs certaines courbes présentent un pic dans les premières itérations et baissent par la suite. Cela pourrait être dû au fait qu'après ces itérations, la politique, devenue performante, devient plus difficile à évaluer.

Une perspective intéressante de ce travail serait de redéfinir sur l'espace d'états-actions les fonctions de base les plus utilisées de la littérature des travaux sur Tetris, notamment celles de Bertsekas et Ioffe (1996), afin d'évaluer plus précisément le succès de LSλPI sur Tetris par rapport aux autres approches. En effet, jusqu'à présent, on ne peut pas comparer directement les résultats de LSPI ou LSλPI sur Tetris avec d'autres approches d'apprentissage par renforcement (Tsitsiklis et van Roy, 1996; Bertsekas et Ioffe, 1996; Kakade, 2001; Farias et van Roy, 2006; Ramon et Driessens, 2004) étant donné que les fonctions de bases proposées par Lagoudakis *et al.* (2002) dans LSPI sont très différentes, et que de plus, elles sont définies sur l'espace d'états-actions.

6.3 Méthode d'entropie croisée

Comme nous l'avons vu dans le chapitre 5, malgré leurs propriétés théoriques intéressantes (notamment l'estimation du score futur), les méthodes d'apprentissage par renforcement ne sont pas celles qui se sont montrées les plus performantes dans la littérature pour optimiser les poids d'un contrôleur. Nous nous intéressons donc ici à la méthode d'entropie croisée appliquée à Tetris par Szita et Lőrincz (2006) et nous revisitons leur approche pour mettre au point un contrôleur plus performant avec d'autres fonctions de base.

En effet, d'une part, les fonctions de base de Dellacherie semblent être les plus compétitives : même

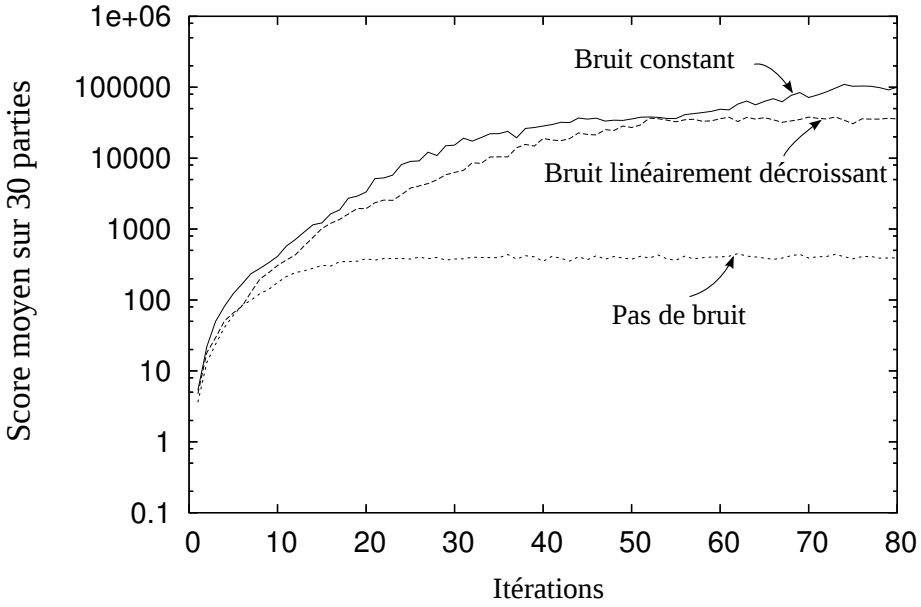


FIGURE 6.2 – Notre implémentation de l'expérience de Szita et Lőrincz (2006). Chaque courbe représente la courbe d'apprentissage moyenne de 10 exécutions pour un type de bruit donné (en échelle logarithmique). Nous observons que l'ajout de bruit améliore significativement les performances. La meilleure des trois courbes d'apprentissage moyennes est celle qui correspond au bruit constant.

avec des poids choisis à la main, le joueur artificiel de Dellacherie donne jusqu'ici les meilleurs résultats. D'autre part, la méthode d'entropie croisée (Szita et Lőrincz, 2006) apparaît comme étant l'algorithme le plus performant pour régler les poids d'un ensemble de fonctions de base donné. Le joueur artificiel que nous mettons au point dans cette section s'appuie sur ces deux observations : nous allons exploiter et compléter les fonctions de base efficaces de Dellacherie, et utiliser la méthode d'entropie croisée pour fixer les poids.

Rappelons que Szita et Lőrincz (2006) ont appliqué la méthode d'entropie croisée avec trois sortes de bruit : un bruit nul, un bruit constant et un bruit linéairement décroissant. Comme le paramètre de bruit semblait avoir un effet crucial sur les résultats, nous avons mené des expériences supplémentaires que nous détaillons maintenant. Szita et Lőrincz (2006) ont exécuté chacune des trois expériences (pas de bruit, bruit constant et bruit décroissant) une seule fois pour des raisons de temps : leurs résultats expérimentaux ont nécessité un mois de calculs. Nous avons apporté un soin tout particulier à l'implémentation de notre simulateur de Tetris, notamment en termes d'optimisation, de manière à pouvoir reproduire leurs expériences plusieurs fois. En effet, nos premiers essais ont montré que plusieurs exécutions de la méthode d'entropie croisée avec les mêmes paramètres pouvaient donner des résultats très différents. Ainsi, nous avons décidé de lancer chacune des trois expériences de Szita et Lőrincz (2006) 10 fois. Avec notre implémentation optimisée de Tetris, cela a pris environ une semaine.

Comme nous avons vu dans la section 5.3 que les scores à Tetris ont une grande variance, il est clair que cette évaluation n'est pas précise. Avec notre implémentation, nous avons essayé d'évaluer chaque vecteur en jouant plus de parties pour voir si c'était un choix crucial, et nous en avons conclu que ce n'était pas le cas. En effet, même si nous avons observé que le nombre d'itérations nécessaires pour atteindre le niveau de performance maximal est inférieur (ce qui est naturel puisque le processus de sélection est plus précis), nous avons remarqué qu'après convergence, les contrôleurs obtenus n'étaient pas meilleurs.

Les résultats que nous avons obtenus sont représentés aux figures 6.2 et 6.3. La figure 6.2 montre pour chaque type de bruit la courbe d'apprentissage moyenne de 10 exécutions (en échelle logarithmique). Nos résultats expérimentaux confirment l'observation de Szita et Lőrincz (2006) selon laquelle ajouter du bruit améliore significativement les résultats. Cependant, nous avons observé que la performance moyenne est meilleure avec le bruit constant. La figure 6.3 montre, pour chaque type de bruit, le détail des 10

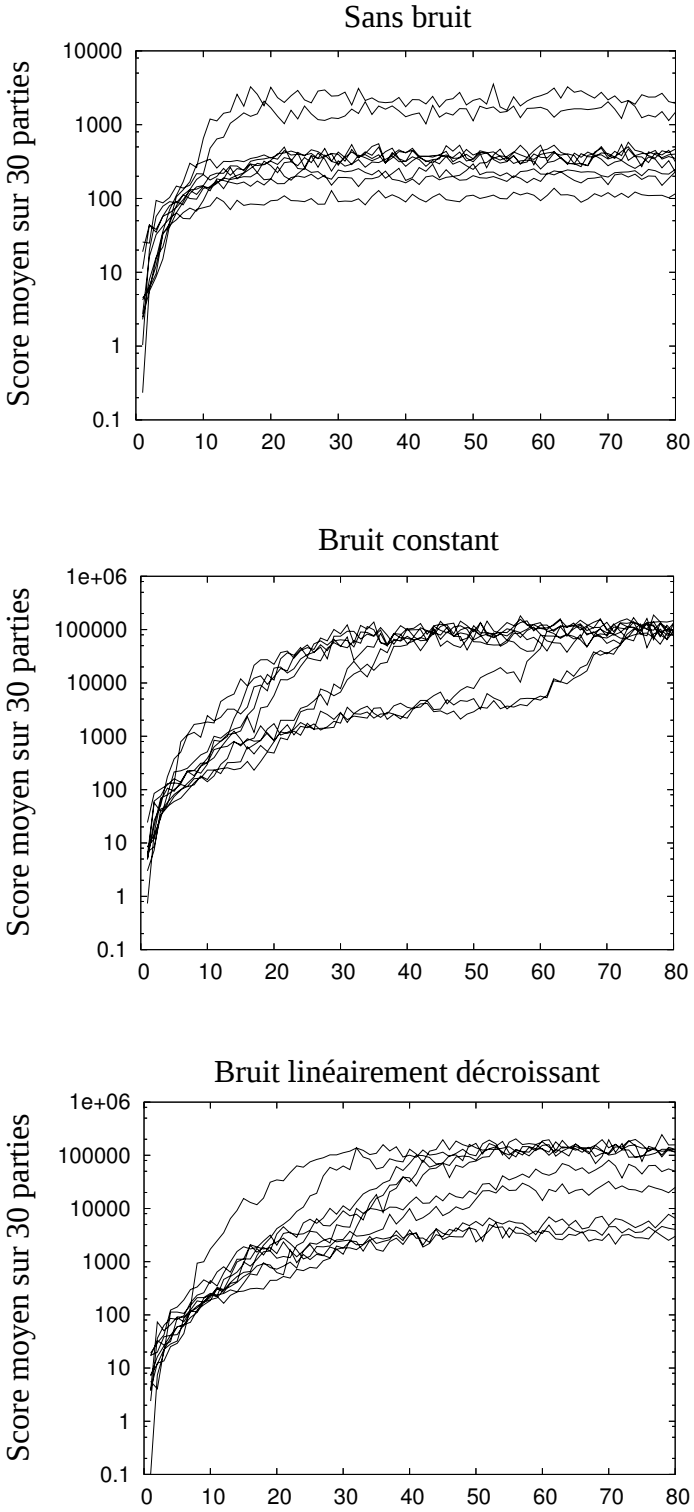


FIGURE 6.3 – Détail des 10 exécutions de chaque expérience de la figure 6.2 (en échelle logarithmique). **Haut (sans bruit) :** la courbe d'apprentissage se stabilise toujours après l'itération 20. Le score moyen atteint varie selon les exécutions (de 100 à 3 000 lignes). **Milieu (bruit constant) :** les 10 exécutions atteignent des performances proches après convergence, entre 100 000 et 200 000 lignes. **Bas (bruit linéairement décroissant) :** les 10 exécutions atteignent des valeurs très différentes, de 5 000 à 250 000 lignes.

exécutions. La meilleure performance est atteinte avec le bruit linéairement décroissant : une des 10 exécutions obtient un contrôleur qui réalise un score moyen de 240 000 ± 37 % lignes. Avec cet intervalle de confiance, nos résultats semblent cohérents avec ceux de Szita et Lőrincz (2006) (350 000 ± 37 %). L'examen de la figure 6.3 donne une meilleure idée sur le choix du bruit : les 10 exécutions avec le bruit constant atteignent toutes des performances similaires après convergence (100 000 à 200 000 ± 37 % lignes), alors qu'avec le bruit décroissant, les performances varient beaucoup entre plusieurs exécutions de la méthode d'entropie croisée. Cela est dû au fait que souvent, le bruit décroissant atteint zéro trop vite et l'algorithme converge avant d'avoir eu le temps de découvrir des bonnes solutions. Par conséquent, si on lance une seule exécution de la méthode d'entropie croisée (c'était le cas dans l'expérience originale de Szita et Lőrincz (2006) et ce sera le cas dans la prochaine section où nous construisons des contrôleurs qui jouent de très longues parties), le bruit constant est plus fiable, à moins de modifier la formule du bruit linéairement décroissant pour le faire diminuer moins vite. Cette conclusion semble aussi indiquer qu'à moins que l'implémentation de Tetris de Szita et Lőrincz (2006) diffère de la nôtre (voir la discussion à la section 5.3.3 sur l'influence significative de paramètres apparemment mineurs), le score de 350 000 ± 37 % lignes par parties pourrait avoir été obtenu avec une part de chance dans la mesure où l'algorithme a été exécuté avec du bruit linéairement décroissant.

6.4 Vers un contrôleur performant

Nous venons de voir que la méthode d'entropie croisée était un algorithme efficace pour optimiser les poids d'un ensemble de fonctions de base à Tetris. Comme nous l'avons vu dans la description des travaux existants au chapitre 5, il est également essentiel de choisir un ensemble de fonctions de base pertinent afin de capturer les aspects importants du jeu de Tetris. Ainsi, une approche naturelle, que nous appliquons dans cette section, est de considérer d'autres fonctions de base de Tetris que les fonctions de Bertsekas et Ioffe (1996).

Nous avons essayé plusieurs combinaisons de fonctions de base, dont celles de Dellacherie puisqu'elles constituent jusqu'ici la meilleure connaissance experte de la littérature pour Tetris. Nous avons aussi introduit deux fonctions de base originales : la *profondeur des trous* et le *nombre de lignes avec trous*. La profondeur des trous indique à quelle distance de la surface du mur se trouvent les trous : c'est la somme du nombre de cellules pleines au-dessus de chaque trou. Le but de cette fonction est d'éviter d'enterrer trop profondément des trous. Notre seconde fonction de base originale compte le nombre de lignes ayant au moins un trou (deux trous sur la même ligne comptent pour un seul).

Nous avons exécuté la méthode d'entropie croisée sur le jeu de taille 10×20 dans les mêmes conditions que Szita et Lőrincz (2006) : nous avons commencé avec une gaussienne centrée à $\mu = (0, 0, \ldots, 0)$ avec variance $\sigma^2 = (100, 100, \ldots, 100)$, nous avons généré $N = 100$ vecteurs à chaque itération, et nous avons sélectionné les 10 meilleurs ($\rho = 10$ %). Conformément aux conclusions de la section précédente, chaque vecteur était évalué en jouant une seule partie et nous avons choisi d'utiliser un bruit constant (avec la même amplitude que Szita et Lőrincz (2006) : $Z_t = 4$). Nous avons lancé l'algorithme avec quatre ensembles de fonctions de base différents : Dellacherie (D), Bertsekas + Dellacherie (BD), Dellacherie + Thiéry (DT), et Bertsekas + Dellacherie + Thiéry (BDT). Comme attendu, les performances obtenues sont nettement meilleures que lorsqu'on se contente des fonctions de base de Bertsekas et Ioffe (1996). Comme les parties sont beaucoup plus longues, nous n'avons lancé qu'une seule exécution pour chacun de ces quatre ensembles de fonctions de base. Bien que notre implémentation soit optimisée, en lançant les quatre expériences sur des machines différentes, ces expériences ont pris un mois.

La figure 6.4 fournit la courbe d'apprentissage pour chacun des quatre ensembles de fonctions de base. Comme dans l'expérience de Szita et Lőrincz (2006), les courbes représentent le score moyen de 30 parties jouées avec le contrôleur moyen généré à la fin de chaque itération. La première observation que l'on peut faire est que nos deux fonctions de base originales ont un impact significatif sur les scores : les courbes correspondantes (les deux courbes en pointillés) sont celles qui réalisent les plus hauts pics. Nous observons également que si l'on supprime les fonctions de base de Bertsekas et Ioffe (1996) (les expériences sans ces fonctions de base correspondent aux deux courbes épaisses), l'algorithme prend moins d'itérations pour converger, ce qui n'est pas surprenant puisqu'il y a moins de paramètres à optimiser, mais atteint des scores similaires. Cela suggère qu'une fois que l'on a les fonctions de base de

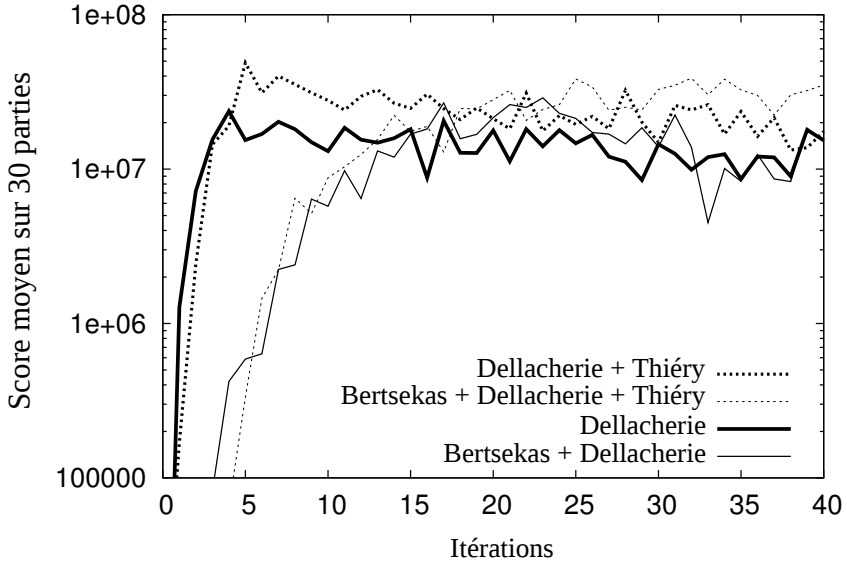


FIGURE 6.4 – Evolution du score moyen de 30 parties (en échelle logarithmique) avec la méthode d'entropie croisée bruitée, avec quatre ensembles de fonctions de base : Dellacherie, Dellacherie + Bertsekas, Dellacherie + Thiéry, Dellacherie + Bertsekas + Thiéry. Les deux courbes qui montent le plus haut sont celles où nos fonctions de base originales sont présentes (ce sont les deux courbes en pointillés). Lorsque les fonctions de base de Bertsekas et Ioffe (1996) ne sont pas présentes (cela correspond aux deux courbes épaisses), l'algorithme converge beaucoup plus vite et les meilleures performances obtenues sont similaires.

Fonctions de base	**DT**	**BDT**	**D**	**BD**
Taille 10×20	35 000 000	36 000 000	17 000 000	20 000 000
Taille 10×16	910 000	910 000	530 000	660 000

TABLE 6.1 – Score moyen du meilleur contrôleur obtenu avec la méthode d'entropie croisée bruitée, pour chaque ensemble de fonctions de base. 100 parties ont été jouées en taille 10×20 (l'intervalle de confiance est 20 %) et 1 600 parties ont été jouées en 10×16 (l'intervalle de confiance est alors de 5 %). Les ensembles de fonctions de base sont représentés par leur première lettre : D pour Dellacherie, B pour Bertsekas, T pour Thiéry. Les meilleurs résultats sont atteints avec les fonctions de base DT et BDT.

Dellacherie, celles de Bertsekas et Ioffe ne donnent pas plus d'informations. On pourra enfin noter que les courbes de la figure 6.4 représentent le score moyen de seulement 30 parties, ce qui fait que l'intervalle de confiance correspondant est assez grand (± 37 %). Pour évaluer les contrôleurs plus précisément, nous avons sélectionné quelques contrôleurs pour chaque ensemble de fonctions de base (nous avons choisi quelques vecteurs de poids correspondant aux pics des courbes de la figure 6.4) et nous les avons fait jouer plus de parties. Le tableau 6.1 reporte, pour le meilleur contrôleur de chaque ensemble de fonctions de base, le score moyen de 100 parties sur un jeu de taille 10×20. L'intervalle de confiance correspondant est de ± 20 %. Nous avons également fait jouer ces mêmes contrôleurs sur un jeu de taille 10×16, pour avoir une borne inférieure sur le score qu'ils réaliseraient sur le Tetris original (voir section 5.3.3). Sur ce jeu de taille réduite, nous avons joué 1 600 parties (les parties étant plus courtes), ce qui donne un intervalle de confiance de 5 %.

L'utilisation de la méthode d'entropie croisée pour optimiser les poids du contrôleur de Dellacherie est pertinente : par rapport aux poids originaux fixés à la main, les poids déterminés automatiquement améliorent significativement les résultats. Les meilleurs scores sont atteints avec les ensembles BDT et DT, qui réalisent des performances équivalentes : $35\ 000\ 000 \pm 20$ % lignes sur le jeu de taille 10×20 et $910\ 000 \pm 5$ % lignes sur le jeu de taille 10×16. Ainsi, ces deux fonctions d'évaluation dépassent l'algorithme qui était jusqu'ici le meilleur à notre connaissance, celui de Dellacherie, qui réalise un score

Fonction de base	**Poids**
Hauteur d'arrivée	-12.63
Erosion	6.60
Transitions de lignes	-9.22
Transitions de colonnes	-19.77
Trous	-13.08
Puits cumulatifs	-10.49
Profondeur des trous	-1.61
Lignes avec trous	-24.04

TABLE 6.2 – Les poids de notre contrôleur DT (Dellacherie + Thiéry). Score moyen : $35\,000\,000 \pm 20$ % lignes en 10×20 et $910\,000 \pm 5$ % en 10×16. Voir section 6.4 et tableau 5.1 pour les définitions des fonctions de base.

moyen de $5\,200\,000 \pm 20$ % sur notre simulateur en 10×20 et $660\,000 \pm 27$ % sur un simulateur du Tetris original (rappelons que jouer en 10×16 sur notre simulateur donne une borne inférieure assez pessimiste sur le résultat avec le Tetris original). Alors que le contrôleur BDT contient 28 fonctions de base, le contrôleur DT n'en possède que 8 : ce dernier est donc plus simple et plus rapide, et il peut ainsi être considéré comme meilleur. Nous donnons ses poids dans le tableau 6.2.

6.5 Reinforcement Learning Competition 2008

En nous basant sur ce travail d'étude de l'état de l'art et d'amélioration des approches les plus performantes, nous avons remporté la Reinforcement Learning Competition 2008 (compétition d'apprentissage par renforcement) dans le domaine de Tetris. Cette compétition faisait jouer des contrôleurs de Tetris sur des instances modifiées du problème, où certaines propriétés du jeu (par exemple la taille de la zone de jeu ou la fonction de récompense) pouvaient varier. Les contrôleurs devaient s'adapter à chaque environnement. La mesure de performance utilisée pour comparer les différents participants était le score total réalisé après un nombre fixé d'interactions, sans pénalité appliquée en fin de partie. Cette mesure de performance avait ainsi beaucoup moins de variance que la mesure naturelle (le score d'une partie) que nous avons considéré dans ce mémoire. Même si le problème était formulé dans le cadre de l'apprentissage par renforcement dans cette compétition, tout type de méthode était autorisé. Nous avons utilisé une version modifiée de notre contrôleur DT présenté plus haut, avec une fonction de base additionnelle appelée « diversité des motifs ». Cette fonction de base, suggérée par Olivier Sigaud (communication personnelle), examine la forme du motif formé par le haut de chaque paire de colonnes voisines et compte combien de motifs différents sont présents. Cela encourage le contrôleur à faire en sorte que le mur puisse accueillir toutes les formes de pièces sans créer de trou. Nous savons que les contrôleurs qui ont obtenu la deuxième et la troisième place (communications personnelles avec Marek Petrik et Istvan Szita respectivement) ont également été mis au point à l'aide de la méthode d'entropie croisée. On peut penser que notre analyse empirique de la méthode d'entropie croisée appliquée à Tetris ainsi que le choix des fonctions de base ont été décisifs pour remporter la compétition.

Conclusion et perspectives

Afin d'appliquer l'apprentissage par renforcement au jeu de Tetris, nous avons résolu de façon exacte une instance réduite du problème (en taille 5×5) avec des outils traditionnels du contrôle optimal, puis nous avons appliqué l'algorithme d'approximation linéaire LSλPI proposé dans le chapitre 4 à la taille normale du jeu (10×20). Nos expériences ont confirmé les résultats du chapitre 5 : par rapport à LSPI (Lagoudakis *et al.*, 2002), LSλPI permet d'améliorer l'efficacité de l'exploitation des échantillons. Des performances similaires sont en effet atteintes en apprenant une base de 1 000 échantillons au lieu de 10 000.

Nous avons par ailleurs revisité l'application de la méthode d'entropie croisée au jeu de Tetris, proposée par Szita et Lőrincz (2006). En reproduisant 10 fois leur expérience d'origine, nous avons approfondi leur analyse expérimentale : en particulier, nous avons observé que le bruit constant semble plus fiable que le bruit linéairement décroissant. En utilisant la connaissance experte (les fonctions de base) de Dellacherie (Fahey, 2003) et deux fonctions de base originales, nous avons construit un contrôleur à une pièce qui donne à notre connaissance les meilleurs résultats à l'heure actuelle et qui a remporté la Reinforcement Learning Competition 2008.

Pour aller plus loin

Pour améliorer encore les expériences, il serait intéressant d'appliquer l'algorithme CMA-ES (Hansen et Ostermeier, 2001) qui est une généralisation de la méthode d'entropie croisée (voir section 5.4). De plus, modifier les formules de bruit ou utiliser d'autres types de bruits (notamment un bruit avec décroissance géométrique) sont des idées à explorer. En effet, nos résultats indiquent que le bruit linéairement décroissant atteint trop vite zéro. On pourrait également aller plus loin en imaginant des fonctions de base plus complexes ou plus expressives. Une première direction naturelle serait d'exploiter d'autres fonctions de base de la littérature (par exemple celles de Xtris (Llima, 2005) ou Fahey (2003)) ou d'en inventer d'autres. Une piste de recherche particulièrement intéressante est le problème de sélectionner et de combiner automatiquement des fonctions de base simples de manière à construire d'autres fonctions de plus haut niveau. Par exemple, de telles combinaisons peuvent faire partie de l'espace de recherche, comme dans la récente approche de Programmation Génétique de Girgin et Preux (2007).

D'une manière générale, un problème significatif concernant le jeu de Tetris est le temps d'exécution nécessaire pour jouer une partie, car les algorithmes ont besoin de jouer de nombreuses parties. C'est encore plus important lorsque les contrôleurs sont meilleurs, car les parties durent de plus en plus longtemps. Nous envisageons plusieurs pistes pour évaluer un contrôleur en réduisant la durée d'une partie ou le nombre de parties jouées.

- Une première idée serait de lancer l'algorithme d'apprentissage sur un jeu de taille réduite. De cette manière, les parties sont plus courtes, on peut donc générer plus de vecteurs, les évaluer plus soigneusement et les itérations peuvent être plus rapides. Cependant, il n'est pas clair qu'un contrôleur obtenu en jouant sur une taille réduite joue ensuite bien sur le jeu standard (10×20). Nous avons effectué quelques expériences, et les performances des contrôleurs construits en apprenant sur des plus petites grilles (comme 10×16) semblent donner des scores légèrement inférieurs à ceux des contrôleurs construits directement avec la grille standard.
- Nous avons vu que pour évaluer la qualité d'un contrôleur, il est préférable de jouer de nombreuses parties. Au lieu de jouer des parties aléatoires, une piste à explorer serait de jouer un petit ensemble de parties prédeterminées, avec des séquences de pièces générées à l'avance. Ainsi, avec cette méthode, on utiliserait les mêmes séquences de pièces pour comparer des contrôleurs différents. Dans la phase de sélection de l'algorithme d'entropie croisée, cela pourrait permettre de sélectionner les meilleurs échantillons de manière plus fiable dans la mesure où on utiliserait la même base de comparaison. Cela dit, il faut que les séquences de pièces prédéterminées soient suffisamment représentatives des parties possibles et il n'est pas clair qu'un contrôleur entraîné sur un ensemble restreint de séquences soit performant sur des parties qu'il n'a jamais rencontrées auparavant.
- Pour réduire le temps nécessaire pour évaluer un contrôleur, une idée prometteuse vient d'une conjecture de Fahey (2003), qui stipule que la durée d'une partie de Tetris (et par conséquent, le nombre de lignes réalisées) peut être estimée à partir des premiers coups joués. En effet, considérons pour chaque hauteur h (c'est-à-dire $h = 0$ à 20) la *fréquence* de h, qui est la proportion du temps où la hauteur du mur a été exactement h pendant les n premiers coups. Avec un bon contrôleur, lorsque h est grand, la fréquence de h est faible puisque les murs hauts apparaissent peu souvent. Fahey a observé expérimentalement qu'avec son contrôleur, lorsque h augmente, la diminution de la fréquence de h est exponentielle. Nous avons fait des expériences qui confirment son observation pour nos contrôleurs. Par conséquent, si l'on estime les paramètres de cette distribution exponentielle (ils diffèrent pour chaque contrôleur) en effectuant une régression, on peut en déduire la fréquence pour $h = 21$, ce qui correspond à la fin de la partie. Pour un contrôleur donné, on pourrait ainsi estimer la durée moyenne d'une partie en jouant seulement n coups au lieu de faire une ou plusieurs parties.

Cependant, nos premières observations indiquent qu'une telle méthode a une grande variance et manque donc de précision, même si l'on joue un grand nombre de coups (de l'ordre de $n = 1\,000\,000$ de coups).

Il reste à approfondir ces pistes afin de réduire le temps d'exécution de l'algorithme.

Une autre question naturelle qui reste posée est de savoir quelle méthode d'optimisation est la plus adaptée pour Tetris. Les travaux de l'état de l'art pour fixer les poids (Llima, 2005; Böhm *et al.*, 2005; Szita et Lőrincz, 2006) ne peuvent pas être comparés directement car ils s'appuient sur différentes implémentations et différentes fonctions de base. De plus, contrairement aux deux autres approches et aux travaux d'apprentissage par renforcement, Böhm *et al.* (2005) considèrent uniquement des contrôleurs à deux pièces, rendant ainsi leur performance inconnue par rapport à celles des contrôleurs à une pièce. Il serait intéressant d'implémenter et d'exécuter ces méthodes dans les mêmes conditions, pour déterminer dans quelles circonstances la méthode d'entropie croisée, à laquelle nous nous sommes intéressés dans cette thèse, peut réaliser de meilleurs résultats que les approches d'optimisation (si c'est le cas), et si de telles observations sont spécifiques à Tetris ou peuvent également s'appliquer à d'autres problèmes.

Conclusion générale

Résumé de la démarche

Cette thèse s'est intéressée à la manière pour un agent informatique d'apprendre automatiquement un comportement (ou politique) à partir d'une série d'expériences. Dans ce contexte, l'apprentissage par renforcement propose un cadre formel et de nombreux outils pour construire des politiques et estimer leur valeur.

Dans un premier temps, nous avons présenté quelques algorithmes fondamentaux du contrôle optimal stochastique dans le cas exact, en particulier λPI (Bertsekas et Ioffe, 1996) qui propose d'itérer sur les politiques de façon optimiste, c'est-à-dire de changer de politique sans attendre d'avoir évalué complètement la politique précédente. Nous avons ensuite abouti à une écriture unifiée de ces algorithmes (Unified Policy Iteration) qui exprime la notion d'optimisme et nous avons montré la convergence de cette proposition. Des expériences sur un problème de type navigation discrète ont permis d'illustrer Unified Policy Iteration et d'étudier l'optimisme.

Par la suite, nous nous sommes intéressés au cas approché, où la fonction de valeur est représentée par une architecture paramétrique et estimée à l'aide d'échantillons. Nous avons d'abord démontré l'existence d'une borne de performance sur les versions approximatives de Unified Policy Iteration. Puis nous avons présenté un état de l'art des méthodes d'apprentissage par renforcement avec approximation linéaire. Parmi ces méthodes, deux approches d'itération sur les politiques ont particulièrement retenu notre attention : LSPI (Lagoudakis et Parr, 2003), qui évalue les politiques de façon off-policy, et AλPI (Bertsekas et Ioffe, 1996), qui les évalue de façon optimiste. Nous avons alors proposé l'algorithme LSλPI qui cumule les avantages de ces deux approches. Nos expériences ont montré que l'usage du paramètre λ peut permettre d'améliorer la qualité de l'estimation et les performances par rapport à LSPI.

Dans une troisième partie, nous avons étudié spécifiquement l'application du jeu de Tetris, un problème à grand espace d'états que tentent de traiter plusieurs approches d'apprentissage par renforcement, mais aussi des méthodes d'optimisation directe de la politique et des algorithmes réglés manuellement. Nous avons dressé le premier état de l'art complet de ces travaux ainsi que la liste des fonctions base employées par chacun d'entre eux. Après avoir souligné la difficulté d'évaluer et de comparer les performances des joueurs artificiels, nous avons mené plusieurs expériences. Nous avons résolu une instance réduite de Tetris de manière exacte avec le contrôle optimal, et dans le cas approché, nous avons confirmé empiriquement que par rapport à LSPI, LSλPI améliore l'efficacité de l'exploitation des échantillons. Nous avons par ailleurs revisité la méthode d'entropie croisée appliquée à Tetris (Szita et Lőrincz, 2006) et montré qu'en y introduisant une meilleure connaissance experte comme celle de Dellacherie (Fahey, 2003), on peut construire un contrôleur qui dépasse les performances de l'état de l'art. Le contrôleur ainsi obtenu nous a permis de remporter l'épreuve de Tetris de la Reinforcement Learning Competition 2008 (compétition d'apprentissage par renforcement).

Optimisme et compromis biais-variance

Tout au long de ce mémoire, dans le cas exact comme dans le cas approché, nous nous sommes posés la question de savoir ce que peut apporter l'idée de changer de politique sans attendre d'avoir évalué complètement la politique précédente (l'optimisme), en particulier via l'utilisation du paramètre λ de λPI (Bertsekas et Ioffe, 1996).

Formellement, dans λPI, l'optimisme consiste à remplacer l'équation de Bellman usuelle $V = T_\pi V$ par une équation de Bellman amortie $V = M_k V$ qui ne caractérise plus la fonction de valeur de la politique

π, mais un pas d'une certaine taille en direction de cette dernière (la taille du pas étant réglable par le paramètre λ). Autrement dit, au lieu de rechercher le point fixe de l'opérateur T_π, on recherche celui de l'opérateur M_k (qui dépend de la politique mais aussi de la fonction de valeur précédente et de λ).

Dans le cas exact, nous avons remplacé T_π par M_k dans Modified Policy Iteration (Puterman, 1994), et constaté que cela n'apportait pas d'amélioration significative en termes de vitesse de convergence. Les propriétés intéressantes de λ apparaissent dans le cas approché, lorsque la fonction de valeur est estimée à l'aide d'un ensemble d'échantillons. En remplaçant T_π par M_k dans l'algorithme LSPI (Lagoudakis et Parr, 2003), nous avons ainsi obtenu un nouvel algorithme d'approximation linéaire (intitulé LSλPI) qui permet d'améliorer l'efficacité de l'estimation grâce cette notion d'optimisme. LSλPI constitue la contribution essentielle de cette thèse, dans la mesure où cet algorithme est le premier qui cumule les caractéristiques suivantes :

- itération sur les politiques optimiste : on change de politique sans attendre d'avoir évalué complètement la politique précédente (via le paramètre λ de λPI),
- compromis biais-variance : le biais introduit par l'optimisme est compensé par une réduction de la variance de l'estimation de la fonction de valeur,
- échantillonnage efficace : il s'agit d'une méthode du second ordre, c'est-à-dire que les informations des échantillons sont exploitées de manière efficace comme dans LSPI, LSTD(λ) (Boyan, 2002) et LSPE(λ) (Nedić et Bertsekas, 2003),
- off-policy : de manière analogue à LSPI, la politique peut être évaluée à partir d'échantillons générées avec une autre politique, et ayant la forme de trajectoires ou non.

LSλPI est une généralisation de LSPI où nous ajoutons la notion d'optimisme afin de réduire la variance des estimations. L'un des intérêts de LSPI est son efficacité pour exploiter les échantillons, grâce à l'approximation du second ordre qu'il effectue et grâce à l'évaluation off-policy (qui permet de réutiliser les mêmes échantillons malgré les changements de politiques). En ajoutant la propriété de réduction de variance à LSPI, nous améliorons encore son efficacité en termes d'échantillons. En pratique, et comme nous l'avons vérifié expérimentalement, cela s'avère utile lorsque le nombre d'échantillons dont on dispose est limité.

Du point de vue de λPI (Bertsekas et Ioffe, 1996), LSλPI peut être vu comme une version approximative où l'on cherche explicitement le point fixe de M_k, ce qui permet d'être off-policy lorsque l'on utilise des fonctions de valeur Q. Les autres approches liées à λPI (comme AλPI (Bertsekas et Ioffe, 1996), LSPE(λ) (Nedić et Bertsekas, 2003) et TD(λ) (Sutton et Barto, 1998)) estiment quant à elles des différences temporelles à partir de trajectoires qui sont on-policy. LSTD(λ) (Boyan, 2002) se base lui aussi sur des trajectoires générées avec la politique à évaluer. Il faut cependant noter le récent travail de Yu (2010) qui propose une version off-policy de LSTD(λ). La politique y est évaluée à l'aide de trajectoires générées avec une autre politique, en faisant appel à des techniques d'*importance sampling*. Il serait particulièrement intéressant d'étudier expérimentalement les performances comparées des deux approches off-policy que sont LSTD(λ) et LSλPI.

L'apprentissage par renforcement, une approche ambitieuse

Lorsqu'ils sont appliqués au problème de Tetris, les travaux d'apprentissage par renforcement ont jusqu'ici moins de succès que des travaux d'optimisation directe de la politique tels que la méthode d'entropie croisée (Szita et Lőrincz, 2006) ou des méthodes évolutionnaires (Llima, 2005). Ce constat est une observation de notre revue de la littérature (chapitre 5), confirmée par les nouveaux résultats que nous avons présentés au chapitre 6.

Cependant, l'apprentissage par renforcement fournit des outils théoriques intéressants pour estimer le score moyen d'un contrôleur (qui est la fonction de valeur) et le score moyen du meilleur contrôleur (la fonction de valeur optimale). Sur une instance réduite du jeu (5×5), l'apprentissage par renforcement est ainsi capable de donner le score moyen optimal depuis tout état. Sur le jeu standard (10×20), même si ces algorithmes souffrent de la dimensionnalité et ont plus de difficultés à être performants lorsqu'ils ont recours à de l'approximation, ils ont le mérite d'estimer le score futur au lieu de se contenter de le maximiser. L'apprentissage par renforcement cherche à construire à la fois une fonction de valeur et une politique, ce qui est un problème plus difficile. Bien que les méthodes d'optimisation comme l'entropie

croisée et les algorithmes évolutionnaires donnent de meilleurs résultats sur Tetris à l'heure actuelle, leur fonction d'évaluation n'a pas de sémantique, elle ne fournit pas d'information sur le score optimal possible.

Un autre avantage des algorithmes d'apprentissage par renforcement est que, dans le cas d'une approximation linéaire du second ordre, les informations données par les échantillons observés sont exploitées de manière efficace. Avec LSPI (Lagoudakis et Parr, 2003), on peut se contenter de générer des échantillons une seule fois et de les réutiliser pour l'évaluation de chaque politique. Notre algorithme LSλPI, tout en conservant cette propriété, améliore encore l'efficacité des échantillons par rapport à LSPI. En effet, en introduisant de l'optimisme dans l'évaluation des politiques, nous réduisons la variance de l'estimation réalisée, et le nombre d'échantillons nécessaires pour obtenir une bonne estimation diminue. À l'inverse, les approches d'exploration de l'espace des politiques telles que la méthode d'entropie croisée et les algorithmes évolutionnaires ont pour principe de générer des centaines de contrôleurs et de les évaluer en générant de grandes quantités d'échantillons. Un modèle génératif est donc nécessaire, ainsi que des ressources plus importantes. Un algorithme tel que LSλPI cherche au contraire à exploiter le plus efficacement possible les informations des échantillons dont il dispose. Cela peut s'avérer particulièrement intéressant dans des applications d'apprentissage en ligne dans lesquelles le nombre d'échantillons disponibles est restreint.

Annexes

Annexe A

Preuve de la borne de performance

Nous détaillons ici la preuve du théorème 1, énoncé page 44, qui donne une garantie sur la performance des algorithmes approchés de type Unified Policy Iteration (voir chapitre 2) sous réserve que l'erreur d'approximation soit bornée à chaque itération.

Théorème 1 (Borne sur la performance de Unified Policy Iteration approché)

Soit un ensemble de poids positifs $(\lambda_i)_{i\geq 1}$ *tels que* $\sum_{i=1}^{\infty} \lambda_i = 1$. *Soit une initialisation quelconque* $\widehat{V}_0$. *Soit un algorithme itératif qui construit la suite* $(\pi_k, \widehat{V}_k)_{k\geq 1}$ *de la manière suivante :*

$$\begin{aligned} \pi_{k+1} &\leftarrow \text{glouton}(\widehat{V}_k) \\ \widehat{V}_{k+1} &\leftarrow \sum_{i=1}^{\infty} \lambda_i (T_{\pi_{k+1}})^i \widehat{V}_k + \epsilon_{k+1}. \end{aligned}$$

ϵ_{k+1} *représente l'erreur d'approximation commise en estimant* V_{k+1}. *Supposons qu'il existe une majoration uniforme* ϵ *de l'erreur : pour tout* k, $\|\epsilon_k\|_\infty \leq \epsilon$. *Alors*

$$\limsup_{k\to\infty} \|V^* - V^{\pi_k}\|_\infty \leq \frac{2\gamma}{(1-\gamma)^2}\epsilon.$$

La preuve de ce théorème, présentée ci-après, est significativement différente de celles qui ont été proposées (séparément) pour les versions approximatives de Value Iteration et Policy Iteration. Dans le cas de Policy Iteration, le raisonnement s'appuie sur la propriété de croissance des fonctions de valeurs, et dans le cas de Value Iteration, il utilise des arguments liés aux contractions. Ces deux types d'arguments ne peuvent pas être utilisés dans le cadre de ce théorème.

Preuve (Scherrer et Thiery, 2010)

Notations et idée générale de la preuve

Nous noterons
- $b_k = \widehat{V}_k - T_{\pi_{k+1}}\widehat{V}_k$ l'erreur de Bellman,
- $d_k = V^* - (\widehat{V}_k - \epsilon_k)$ la différence entre la fonction de valeur optimale et l'itéré $\widehat{V}_k$ (avant erreur),
- $s_k = \widehat{V}_k - \epsilon_k - V^{\pi_k}$ la différence entre l'itéré $\widehat{V}_k$ (avant erreur) et la (vraie) valeur de la politique π_k,
- $\beta = \sum_{n\geq 1} \lambda_n \gamma^n$ (on pourra remarquer que $0 \leq \beta \leq \gamma$).

La distance entre la valeur de la politique optimale et la valeur de la politique courante peut s'écrire de la manière suivante :

$$\begin{aligned}\|V^* - V^{\pi_k}\|_\infty &= \max(V^* - V^{\pi_k}) \\ &= \max(V^* - \widehat{V}_k + \epsilon_k + \widehat{V}_k - \epsilon_k - V^{\pi_k}) \\ &= \max(d_k + s_k) \\ &\leq \max d_k + \max s_k \end{aligned} \tag{A.1}$$

L'idée de la preuve est de calculer des majorations de d_k et de s_k. Comme nous allons le voir dans le détail, les majorations que nous obtiendrons dépendront toutes deux d'une majoration de l'erreur de Bellman b_k, que nous commençons par calculer.

Une borne supérieure sur l'erreur de Bellman b_k Comme π_{k+1} est la politique gloutonne par rapport à $\widehat{V}_k$, on a $T_{\pi_k}\widehat{V}_k \leq T_{\pi_{k+1}}\widehat{V}_k$, ce qui nous permet de dire que

$$\begin{aligned} b_k &= \widehat{V}_k - T_{\pi_{k+1}}\widehat{V}_k \\ &= \widehat{V}_k - T_{\pi_k}\widehat{V}_k + T_{\pi_k}\widehat{V}_k - T_{\pi_{k+1}}\widehat{V}_k \\ &\leq \widehat{V}_k - T_{\pi_k}\widehat{V}_k \\ &= (\widehat{V}_k - \epsilon_k + \epsilon_k) - T_{\pi_k}(\widehat{V}_k - \epsilon_k + \epsilon_k) \\ &= (\widehat{V}_k - \epsilon_k) - T_{\pi_k}(\widehat{V}_k - \epsilon_k) + \epsilon_k - \gamma P_{\pi_k}\epsilon_k \\ &= \sum_{n\geq 1}\lambda_n\left[(T_{\pi_k})^n\widehat{V}_{k-1}\right] - \sum_{n\geq 1}\lambda_n\left[(T_{\pi_k})^{n+1}\widehat{V}_{k-1}\right] + (I - \gamma P_{\pi_k})\epsilon_k \\ &= \sum_{n\geq 1}\lambda_n\left[(T_{\pi_k})^n\widehat{V}_{k-1} - (T_{\pi_k})^{n+1}\widehat{V}_{k-1}\right] + (I - \gamma P_{\pi_k})\epsilon_k \\ &= \sum_{n\geq 1}\lambda_n(\gamma P_{\pi_k})^n(\widehat{V}_{k-1} - T_{\pi_k}\widehat{V}_{k-1}) + (I - \gamma P_{\pi_k})\epsilon_k \\ &= \sum_{n\geq 1}\lambda_n(\gamma P_{\pi_k})^n b_{k-1} + (I - \gamma P_{\pi_k})\epsilon_k. \end{aligned}$$

En utilisant le fait que P_{π_k} est une matrice stochastique, on en déduit :

$$\max b_k \leq \sum_{n\geq 1}\lambda_n\gamma^n \max b_{k-1} + (1+\gamma)\epsilon = \beta \max b_{k-1} + (1+\gamma)\epsilon.$$

On en déduit par récurrence que

$$\max b_k \leq \sum_{j=0}^{k-1}\beta^j(1+\gamma)\epsilon + \beta^k \max b_0 = \frac{1+\gamma}{1-\beta}\epsilon + O(\gamma^k). \tag{A.2}$$

Une borne supérieure sur d_k Etudions à présent le terme d_k et son évolution.

$$\begin{aligned} d_{k+1} &= V^* - (\widehat{V}_{k+1} - \epsilon_{k+1}) \\ &= V^* - \sum_{n\geq 1}\lambda_n(T_{\pi_{k+1}})^n\widehat{V}_k \\ &= \sum_{n\geq 1}\lambda_n\left[V^* - (T_{\pi_{k+1}})^n\widehat{V}_k\right]. \end{aligned} \tag{A.3}$$

Comme π_{k+1} est la politique gloutonne par rapport à $\widehat{V}_k$, on a $T_{\pi^*}\widehat{V}_k \leq T_{\pi_{k+1}}\widehat{V}_k$, et donc

$$\begin{aligned}
V^* - (T_{\pi_{k+1}})^n \widehat{V}_k &= T_{\pi^*}V^* - T_{\pi^*}\widehat{V}_k + T_{\pi^*}\widehat{V}_k - T_{\pi_{k+1}}\widehat{V}_k + T_{\pi_{k+1}}\widehat{V}_k - \\
&\quad -(T_{\pi_{k+1}})^2\widehat{V}_k + (T_{\pi_{k+1}})^2\widehat{V}_k - \ldots + (T_{\pi_{k+1}})^{n-1}\widehat{V}_k - (T_{\pi_{k+1}})^n\widehat{V}_k \\
&\leq T_{\pi^*}V^* - T_{\pi^*}\widehat{V}_k + \gamma P_{\pi_{k+1}}(\widehat{V}_k - T_{\pi_{k+1}}\widehat{V}_k) + \\
&\quad +(\gamma P_{\pi_{k+1}})^2(\widehat{V}_k - T_{\pi_{k+1}}\widehat{V}_k) + \ldots + (\gamma P_{\pi_{k+1}})^{n-1}(\widehat{V}_k - T_{\pi_{k+1}}\widehat{V}_k) \\
&= \gamma P_{\pi^*}(V^* - \widehat{V}_k) + \\
&\quad + \left[\gamma P_{\pi_{k+1}} + (\gamma P_{\pi_{k+1}})^2 + \ldots + (\gamma P_{\pi_{k+1}})^{n-1}\right](\widehat{V}_k - T_{\pi_{k+1}}\widehat{V}_k) \\
&= \gamma P_{\pi^*}(V^* - (\widehat{V}_k - \epsilon_k)) - \gamma P_{\pi^*}\epsilon_k + \\
&\quad + \left[\gamma P_{\pi_{k+1}} + (\gamma P_{\pi_{k+1}})^2 + \ldots + (\gamma P_{\pi_{k+1}})^{n-1}\right](\widehat{V}_k - T_{\pi_{k+1}}\widehat{V}_k) \\
&= \gamma P_{\pi^*}d_k - \gamma P_{\pi^*}\epsilon_k + \left[\gamma P_{\pi_{k+1}} + (\gamma P_{\pi_{k+1}})^2 + \ldots + (\gamma P_{\pi_{k+1}})^{n-1}\right] b_k.
\end{aligned}$$

Comme P_{π^*} et $P_{\pi_{k+1}}$ sont des matrices stochastiques, on en déduit

$$\begin{aligned}
\max[V^* - (T_{\pi_{k+1}})^n \widehat{V}_k] &\leq \gamma \max d_k + \gamma\epsilon + (\gamma + \gamma^2 + \ldots + \gamma^{n-1}) \max b_k \\
&= \gamma \max d_k + \gamma\epsilon + \frac{\gamma - \gamma^n}{1-\gamma} \max b_k.
\end{aligned}$$

En utilisant l'équation (A.3), on obtient la récurrence suivante sur $\max d_k$:

$$\max d_{k+1} \leq \gamma \max d_k + \gamma\epsilon + \sum_{n\geq 1} \lambda_n \left[\frac{\gamma - \gamma^n}{1-\gamma} \max b_k\right] = \gamma \max d_k + \gamma\epsilon + \frac{\gamma - \beta}{1-\gamma} \max b_k.$$

A l'aide de la majoration de l'erreur de Bellman obtenue précédemment (équation (A.2)) on en déduit :

$$\max d_{k+1} \leq \gamma \max d_k + \gamma\epsilon + \frac{\gamma - \beta}{(1-\gamma)(1-\beta)}(1+\gamma)\epsilon + O(\gamma^k),$$

ce qui donne, en prenant la limite supérieure,

$$\limsup_{k\to\infty} \max d_k \leq \frac{\gamma}{1-\gamma}\epsilon + \left[\frac{\gamma - \beta}{(1-\gamma)^2(1-\beta)}\right](1+\gamma)\epsilon. \tag{A.4}$$

Une borne supérieure sur s_k Considérons maintenant le terme s_k de l'équation (A.1) :

$$\begin{aligned}
s_{k+1} &= \widehat{V}_{k+1} - \epsilon_{k+1} - V^{\pi_{k+1}} \\
&= \sum_{n\geq 1} \lambda_n \left[(T_{\pi_{k+1}})^n \widehat{V}_k\right] - (T_{\pi_{k+1}})^\infty \widehat{V}_k \\
&= \sum_{n\geq 1} \lambda_n \left[(T_{\pi_{k+1}})^n \widehat{V}_k - (T_{\pi_{k+1}})^\infty \widehat{V}_k\right].
\end{aligned} \tag{A.5}$$

On peut observer que

$$\begin{aligned}
(T_{\pi_{k+1}})^n \widehat{V}_k - (T_{\pi_{k+1}})^\infty \widehat{V}_k &= (T_{\pi_{k+1}})^n \widehat{V}_k - (T_{\pi_{k+1}})^{n+1}\widehat{V}_k + (T_{\pi_{k+1}})^{n+1}\widehat{V}_k - (T_{\pi_{k+1}})^{n+2}\widehat{V}_k + \ldots \\
&= (\gamma P_{\pi_{k+1}})^n(\widehat{V}_k - T_{\pi_{k+1}}\widehat{V}_k) + (\gamma P_{\pi_{k+1}})^{n+1}(\widehat{V}_k - T_{\pi_{k+1}}\widehat{V}_k) + \ldots \\
&= (\gamma P_{\pi_{k+1}})^n [I + \gamma P_{\pi_{k+1}} + (\gamma P_{\pi_{k+1}})^2 + \ldots] b_k.
\end{aligned}$$

Comme précédemment, en utilisant le fait que $P_{\pi_{k+1}}$ est une matrice stochastique, on obtient :

$$\max[(T_{\pi_{k+1}})^n \widehat{V}_k - (T_{\pi_{k+1}})^\infty \widehat{V}_k] \leq \gamma^n(1 + \gamma + \gamma^2 + \ldots) \max b_k = \frac{\gamma^n}{1-\gamma} \max b_k.$$

En utilisant l'équation (A.5), on en déduit une majoration de $\max s_{k+1}$:

$$\max s_{k+1} \leq \frac{1}{1-\gamma}\left[\sum_{n\geq 1}\lambda_n\gamma^n \max b_k\right] = \frac{\beta}{1-\gamma}\max b_k.$$

A l'aide de la majoration de l'erreur de Bellman (équation (A.2)) et en prenant la limite supérieure, on a

$$\limsup_{k\to\infty}\max s_k = \frac{\beta}{(1-\gamma)(1-\beta)}(1+\gamma)\epsilon. \tag{A.6}$$

Conclusion de la preuve Finalement, revenons à l'équation (A.1) et utilisons les majorations que nous venons d'obtenir pour d_k (équation (A.4)) et s_k (équation (A.6)) :

$$\begin{aligned}
\limsup_{k\to\infty}\|V^* - V^{\pi_k}\|_\infty &\leq \limsup_{k\to\infty}\max d_k + \limsup_{k\to\infty}\max s_k \\
&= \frac{\gamma}{1-\gamma}\epsilon + \left[\frac{\gamma-\beta}{(1-\gamma)^2(1-\beta)} + \frac{\beta}{(1-\gamma)(1-\beta)}\right](1+\gamma)\epsilon. \\
&= \frac{\gamma}{1-\gamma}\epsilon + \left[\frac{\gamma-\beta+(1-\gamma)\beta}{(1-\gamma)^2(1-\beta)}\right](1+\gamma)\epsilon. \\
&= \frac{\gamma}{1-\gamma}\epsilon + \left[\frac{\gamma}{(1-\gamma)^2}\right](1+\gamma)\epsilon. \\
&= \frac{\gamma(1-\gamma)+\gamma(1+\gamma)}{(1-\gamma)^2}\epsilon \\
&= \frac{2\gamma}{(1-\gamma)^2}\epsilon. \quad \blacksquare
\end{aligned}$$

Bibliographie

Andrew, A. M. (2000). An introduction to support vector machines and other kernel-based learning methods. *Robotica*, 18(6):687–689.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ.

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

Bertsekas, D. et Tsitsiklis, J. (1996). *Neurodynamic Programming*. Athena Scientific.

Bertsekas, D. P., Borkar, V. S. et Nedi'c, A. (2003). Improved temporal difference methods with linear function approximation. Rapport technique, MIT.

Billingsley, P. (1995). *Probability and measure*. John Wiley & Sons, New York, 3ème édition.

Bishop, C. M. (1996). *Neural Networks for Pattern Recognition*. Oxford University Press, USA, 1ère édition.

Böhm, N., Kókai, G. et Mandl, S. (2005). An Evolutionary Approach to Tetris. *In The Sixth Metaheuristics International Conference (MIC2005)*.

Boyan, J. A. (2002). Technical update : Least-squares temporal difference learning. *Machine Learning*, 49:233–246.

Bradtke, S. J. et Barto, A. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22:33–57.

Breiman, L., Friedman, J., Olshen, R. et Stone, C. (1984). *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA.

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

Carr, D. (2005). Applying reinforcement learning to tetris. Rapport technique, Computer Science department of Rhodes University.

de Boer, P., Kroese, D., Mannor, S. et Rubinstein, R. (2004). A tutorial on the cross-entropy method. *Annals of Operations Research*, 1(134):19–67.

Demaine, E. D., Hohenberger, S. et Liben-Nowell, D. (2003). Tetris is hard, even to approximate. *In Proc. 9th International Computing and Combinatorics Conference (COCOON 2003)*, pages 351–363.

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

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

Girgin, S. et Preux, P. (2007). Feature discovery in reinforcement learning using genetic programming. Rapport technique RR-6358, INRIA.

Golub, G. H. et Loan, C. F. V. (1996). *Matrix Computations*. The Johns Hopkins University Press, 3rd édition.

Hansen, N. et Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195.

Hyvärinen, A. (2001). Independent component analysis. *Neural Computing Surveys*, 2.

Kakade, S. (2001). A natural policy gradient. *In Advances in Neural Information Processing Systems (NIPS 14)*, pages 1531–1538.

Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R. et Wu, A. Y. (2002). An efficient k-means clustering algorithm : Analysis and implementation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(7):881–892.

Kearns, M. et Singh, S. (2000). Bias-variance error bounds for temporal difference updates. *In Proceedings of the 13th Annual Conference on Computational Learning Theory*, pages 142–147.

Kohonen, T. (1989). *Self-organization and associative memory*. Springer-Verlag New York, Inc, New York, NY, USA.

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

Lagoudakis, M. G., Parr, R. et Littman, M. L. (2002). Least-squares methods in reinforcement learning for control. *In SETN'02 : Proceedings of the Second Hellenic Conference on AI*, pages 249–260. Springer-Verlag.

Llima, R. E. (2005). Xtris readme. `http://www.iagora.com/~espel/xtris/README`.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, New York.

Munos, R. (2003). Error bounds for approximate policy iteration. *In ICML'03 : Proceedings of the 20th international conference on Machine learning*, pages 560–567.

Munos, R. (2004). Algorithme d'itération sur les politiques avec approximation linéaire. *Journal Electronique d'Intelligence Artificielle*, 1:4–37.

Nedić, A. et Bertsekas, D. P. (2003). Least squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems*, 13(1-2):79–110.

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

Ramon, J. et Driessens, K. (2004). On the numeric stability of gaussian processes regression for relational reinforcement learning. *In ICML-2004 Workshop on Relational Reinforcement Learning*, pages 10–14.

Ritter, H., Martinetz, T. et Schulten, K. (1992). *Neural Computation and Self-Organizing Maps ; An Introduction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Russell, S. J., Norvig, P., Candy, J. F., Malik, J. M. et Edwards, D. D. (1996). *Artificial intelligence : a modern approach*. Prentice-Hall, Inc.

Scherrer, B. (2007). Performance Bounds for Lambda Policy Iteration. Rapport technique, INRIA.

Scherrer, B. et Thiery, C. (2010). Performance bound for approximate optimistic policy iteration. Rapport technique, Loria - INRIA.

Schoknecht, R. (2002). Optimality of reinforcement learning algorithms with linear function approximation. *In Advances in Neural Information Processing Systems (NIPS 15)*, pages 1555–1562.

Sutton, R. et Barto, A. (1998). *Reinforcement Learning, An introduction*. Bradford Book. The MIT Press.

Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C. et Wiewiora, E. (2009). Fast gradient-descent methods for temporal-difference learning with linear function approximation. *In ICML'09 : Proceedings of the 26th Annual International Conference on Machine Learning*, pages 993–1000.

Szepesvári, C. et Munos, R. (2005). Finite time bounds for sampling based fitted value iteration. *In ICML'05 : Proceedings of the 22nd international conference on Machine learning*, pages 880–887. ACM.

Szita, I. et Lőrincz, A. (2006). Learning Tetris using the noisy cross-entropy method. *Neural Computation*, 18(12):2936–2941.

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

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

Thiery, C. et Scherrer, B. (2009c). Une approche modifiée de Lambda-Policy Iteration. *In Journées Francophones Planification Décision Apprentissage*, Paris France. UPMC-Paris 6.

Thiery, C. et Scherrer, B. (2010). Least-Squares λ Policy Iteration : Bias-Variance Trade-off in Control Problems. *In ICML'10 : Proceedings of the 27th Annual International Conference on Machine Learning*.

Tsitsiklis, J. N. et Roy, B. V. (1997). An analysis of temporal-difference learning with function approximation. Rapport technique, IEEE Transactions on Automatic Control.

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

Yu, H. (2010). Convergence of least squares temporal difference methods under general conditions. *In ICML'10 : Proceedings of the 27th Annual International Conference on Machine Learning*.

Yu, H. et Bertsekas, D. P. (2009). Convergence results for some temporal difference methods based on least squares. *IEEE Trans. Automatic Control*, 54:1515–1531.