De l'échantillonnage adaptatif à la résolution de jeux

Nathanaël Fijalkow, Emilie Kaufmann

Chapitre 2

De l'échantillonnage adaptatif à la résolution de jeux

Nathanaël Fijalkow, Emilie Kaufmann

Dans ce chapitre nous introduisons le problème des bandits, qui est la brique de base de l'apprentissage par renforcement. L'objectif est de présenter l'algorithme de Monte Carlo Tree Search (MCTS) pour la résolution de jeux, et plus précisément la variante UCT.

2.1 Introduction

L'apprentissage par renforcement [21] est un domaine de recherche actif où l'objectif est d'entraîner un agent (par exemple un programme jouant à un jeu) à résoudre une tâche (gagner le jeu) en procédant par essai/erreur lors d'interactions répétées avec son environnement. A chacune de ces interactions, l'agent doit sélectionner une action, qui fait évoluer son état (la position du plateau) et peut donner lieu à une récompense instantanée, l'objectif de l'agent étant de maximiser une notion de récompense *cumulée*. L'environnement peut être stochastique (aléa lié à des lancers de dés) et une formalisation mathématique souvent adoptée pour l'apprentissage par renforcement est celle de la résolution d'un Processus de Décision Markovien (MDP pour Markov Decision Process [16]) de paramètres inconnus.

L'apprentissage par renforcement a de nombreuses applications potentielles pour le contrôle de systèmes complexes (robotique, gestion d'une smart grid, conduite autonome d'un véhicule) mais ses plus grands succès ont été pour l'instant obtenus dans le domaine de la résolution de jeux.

Dans les années 1990, une première IA utilisant des techniques d'apprentissage par renforcement surpasse les meilleurs joueurs humains pour le jeu de Backgammon [22]. Dès les années 2000, de tels outils ont conduit à la proposition de premières IA pour le jeu de Go [5] et en 2015 le programme AlphaGo de DeepMind bat les meilleurs joueurs mondiaux [19]. Il est suivi de peu par AlphaGo Zero [20] qui, au contraire de son prédécesseur, n'utilise plus de base de données de parties jouées par des experts humains et atteint des performances supérieures.

Bien qu'étant un jeu déterministe, le Go est extrêmement complexe de par la taille de l'espace d'actions possibles (position d'une pierre sur le goban, une grille de taille 19 par 19). Pour choisir l'action suivante dans une position donnée du goban, il n'est donc pas possible de développer l'arbre de toutes les séquences d'actions successives à partir de cette position et les IA de Go sont basées sur une exploration *adaptative* d'une version tronquée de cet arbre de jeu. Dans ce chapitre, nous allons présenter en détail ces algorithmes d'exploration adaptative d'arbre, appelées Monte Carlo Tree Search (MCTS) en anglais, ainsi que leur application à la résolution de jeux. Un des algorithmes de MCTS les plus populaires, appelé UCT [12] est basé sur un *algorithme de bandit*.

Afin de poser les bases nécessaires à la description des algorithmes de MCTS, la première partie de ce chapitre consiste en une présentation des problèmes de bandits et de leur résolution. Dans un modèle de bandit, un agent échantillonne de manière adaptative différentes distributions de probabilités, appelées bras en référence au bras d'un bandit manchot (ancien nom pour une machine à sous). Sa stratégie d'échantillonage dépend bien évidemment de l'objectif à atteindre, qui peut être lié ou non à la maximisation de récompenses, mais elle est toujours basée sur des outils statistiques comme la construction d'intervalles de confiance.

Dans une seconde partie nous faisons une courte introduction à l'apprentissage par renforcement dans le cadre des MDPs, et présentons l'algorithme de MCTS, ou plutôt la variante UCT de la famille d'algorithmes MCTS. Nous expliquons ensuite comment cet algorithme s'étend au cadre des jeux à somme nulle.

2.2 Problèmes de bandits à plusieurs bras

Dans un modèle de bandit à plusieurs bras, un agent interagit avec une collection de K distributions de probabilités (les bras), notées $\nu_1, \dots, \nu_K$ ayant des caractéristiques initialement inconnues de l'agent. En particulier, on fait l'hypothèse que chaque distribution ν_a possède une espérance et

on note μ_a l'espérance de la loi ν_a. À chaque instant t, l'agent sélectionne un bras $A_t \in [K]$ [1] et observe ensuite un échantillon X_t de la loi associée : $X_t \sim \nu_{A_t}$. La stratégie de l'agent (appelée parfois algorithme de bandit) est séquentielle au sens ou le bras sélectionné à l'instant t ne peut dépendre que des bras choisis précédemment, $A_1, \ldots, A_{t-1}$ et des échantillons observés $X_1, \ldots, X_{t-1}$ (et éventuellement d'un aléa indépendant pour les stratégies utilisant de la randomisation).

Différents objectifs ont été considérés dans la littérature (voir par exemple [14] pour une présentation récente du domaine). Le problème de bandit le plus étudié est un problème particulier d'apprentissage par renforcement où les échantillons X_t collectés par l'agent sont perçus comme des récompenses dont on cherche à maximiser la somme totale, $\mathbb{E}[\sum_{t=1}^{T} X_t]$. Dans le cadre général de l'apprentissage par renforcement, le choix d'une action (un bras) génère une récompense mais conduit aussi à l'évolution de l'état de l'agent. Dans le cas particulier du bandit, il n'y a qu'un seul état et l'agent fait face de manière répétée aux mêmes actions. Toutefois, ce problème simple nécessite déjà de réaliser un compromis entre l'exploration de l'environnement (estimation des distributions inconnues des bras) et l'exploitation (échantillonner plus les bras qui semblent meilleurs). Une autre partie de la littérature considère des problèmes dit d'exploration pure [3, 6], où la contrainte de maximiser les récompenses est absente, mais où l'on cherche à répondre le plus rapidement possible à une question concernant les bras, par exemple, quel bras a la moyenne la plus grande ? Nous présenterons un algorithme pour résoudre ce problème d'identification du meilleur bras dans la partie 2.2.2, mais nous nous concentrerons d'abord dans la partie 2.2.1 sur des algorithmes maximisant les récompenses.

Avant de décrire plus formellement ces deux objectifs, nous pouvons nous poser la question de l'intérêt des modèles de bandits. Comme nous l'avons dit dans l'introduction, les algorithmes de bandits peuvent servir de brique de base pour la construction des algorithmes de MCTS, mais ils ont aussi été étudiés pour beaucoup d'autres applications. Dès les années 1950, les modèles de bandits sont présentés notamment comme une modélisation très simple des essais cliniques séquentiels [23, 17]. Dans ce contexte, chaque bras modélise l'efficacité d'un traitement possible pour une maladie donnée : ν_a est une variable aléatoire de Bernoulli de paramètre μ_a, qui est dans ce cas la probabilité que le traitement soit efficace.

Pour le t-ème patient de l'essai clinique, le médecin sélectionne un traitement A_t parmi K traitements possibles (d'efficacité inconnue au début de l'essai) en exploitant possiblement les résultats observés sur les patients

1. pour tout entier naturel $n \in \mathbb{N}^*$, on note $[n] = \{1, \ldots, n\}$.

$\mathcal{B}(\mu_1)$ $\mathcal{B}(\mu_2)$ $\mathcal{B}(\mu_3)$ $\mathcal{B}(\mu_4)$ $\mathcal{B}(\mu_5)$

FIGURE 2.1 – *Un essai clinique avec cinq traitements où l'efficacité du traitement a est modélisé par une variable aléatoire de Bernoulli* $\mathcal{B}(\mu_a)$.

précédents. Il observe ensuite un indicateur $X_t = 1$ si le traitement est un succès et $X_t = 0$ si c'est un échec. Maximiser les récompenses dans ce modèle revient à maximiser le nombre de patients guéris pendant l'essai clinique. Les essais cliniques étant rarement à visée thérapeutique, le problème d'identification du meilleur bras (trouver le traitement d'efficacité maximale dans un essai aussi court que possible) fait également sens pour cette application. Cette modélisation ne prenant pas en compte les caractéristiques de patients et supposant que l'efficacité s'observe immédiatement après l'administration du traitement est évidemment simpliste et les algorithmes de bandits ont jusqu'alors été peu utilisés pour les essais cliniques [18]. En revanche, ils ont été beaucoup utilisés pour des applications moins sensibles liés à l'optimisation de contenu web, comme la publicité en ligne (maximiser le nombre de clics en choisissant de manière adaptative quelle publicité afficher pour chaque utilisateur) ou les systèmes de recommandations [15].

2.2.1 L'algorithme UCB pour maximiser ses récompenses

Pour maximiser la somme de ses récompenses, l'agent va chercher à échantillonner autant que possible le meilleur bras défini par $a_\star := \operatorname{argmax}_a \mu_a$ de moyenne $\mu_\star = \max_a \mu_a$. La performance d'une stratégie sera mesurée par son *regret*, qui quantifie la différence de performance entre la stratégie oracle jouant uniquement le bras $a_\star$ et la stratégie de l'agent. Dans un modèle de bandit $\nu = (\nu_1, \dots, \nu_K)$, le regret d'une stratégie

$\mathcal{A} = (A_t)_{t\in\mathbb{N}}$ au bout du temps T est défini par

$$\begin{aligned}\mathcal{R}_\nu(\mathcal{A}, T) &= T\mu_\star - \mathbb{E}_\nu\left[\sum_{t=1}^{T} X_t\right] \\ &= \mathbb{E}_\nu\left[\sum_{t=1}^{T}(\mu_\star - \mu_{A_t})\right]\end{aligned}$$

où la deuxième égalité est obtenue en utilisant le fait que l'espérance de X_t conditionnellement à A_t est μ_{A_t}. On va donc chercher à construire des stratégies minimisant le regret qui, de manière équivalente, maximisent l'espérance de la somme des récompenses. Notons que l'horizon T utilisé dans la définition du regret peut ou non être connu de l'agent. On s'intéressera en particulier aux algorithmes appelés en anglais *anytime* qui ne dépendent pas de T et dont on peut majorer le regret pour n'importe quel horizon T.

Pour comprendre ce qu'est un bon algorithme minimisant le regret, on observe que le regret peut se réécrire en fonction du nombre de sélections de chaque bras par l'algorithme. En introduisant pour tout t la variable aléatoire $N_a(t) = \sum_{s=1}^{t} \mathbb{1}(A_s = a)$ qui compte le nombre de fois où l'agent a choisi le bras a lors des t premiers tours de jeu, on peut écrire

$$\mathcal{R}_\nu(\mathcal{A}, T) = \sum_{a=1}^{K}(\mu_\star - \mu_a)\mathbb{E}_\nu\left[N_a(T)\right]$$

Pour avoir un regret faible, un algorithme doit donc peu souvent sélectionner les bras sous-optimaux, pour lesquels le *gap* $\Delta_a := \mu_\star - \mu_a$ est positif et ce d'autant moins que ce gap est grand. Les moyennes des bras étant inconnues, cela nécessite de réaliser un minimum d'*exploration* (essai des différents bras pour identifier les bons et les mauvais bras) tout en *exploitant* les bras qui paraissent les meilleurs.

L'objectif est de construire des algorithmes dont le regret est au moins sous-linéaire : $\mathcal{R}_\nu(\mathcal{A}, T) = o(T)$ lorsque T tend vers l'infini, ce qui garantit que la récompense moyenne collectée converge vers $\mu_\star$. Nous allons voir que, sous certaines hypothèses sur les distributions des bras, il est même possible d'obtenir un regret logarithmique.

Pourquoi l'estimation n'est pas suffisante Pour estimer la qualité d'un bras après t tours de jeu, une idée naturelle est de calculer la récompense moyenne que ce bras a donné jusqu'alors, définie par

$$\widehat{\mu}_a(t) = \frac{1}{N_a(t)}\sum_{s=1}^{t}\mathbb{1}(A_s = a)X_s \quad \text{si} \quad N_a(t) > 0$$

et $\widehat{\mu}_a(t) = +\infty$ si $N_a(t) = 0$. Un agent faisant confiance à la qualité de ces estimations choisirait alors à l'instant $t+1$

$$A_{t+1}^{\text{greedy}} = \operatorname*{argmax}_{a \in [K]} \widehat{\mu}_a(t).$$

Cette stratégie gloutonne (greedy en anglais) appelée parfois *Follow the Leader* conduit toutefois en général à un regret linéaire.

Pour le comprendre, prenons l'exemple d'un modèle de bandit à deux bras où la moyenne du bras a est une distribution de Bernoulli de moyenne μ_a avec $\mu_1 > \mu_2$. Du fait de l'initialisation des moyennes empiriques, la stratégie greedy va commencer par sélectionner une fois chaque bras. Si lors de cette initialisation le bras 1 donne une récompense 0 et le bras 2 une récompense 1 (ce qui arrive avec probabilité $(1-\mu_1).\mu_2$), on aura toujours $\mu_2(t) > \mu_1(t)$ pour $t \geqslant 2$ ce qui conduit à $N_1(T) = 1$ pour $N_2(T) = T-1$. On a alors

$$\mathcal{R}_\nu(\text{greedy}, T) \geqslant (1-\mu_1)\mu_2(\mu_1 - \mu_2)(T-1).$$

Ainsi, cette stratégie d'exploitation pure ne fonctionne pas pour minimiser le regret dans un modèle de bandit. A l'inverse, une stratégie explorant uniformément les bras (pour laquelle $N_a(T) = T/K$) aurait également un regret linéaire. Une bonne stratégie doit donc *mélanger exploration et exploitation*.

Une première idée pour faire cela (et corriger le défaut noté ci-dessus pour la stratégie gloutonne) consiste à effectuer une phase d'exploration où chaque bras est tiré m fois, avec cette fois $m > 1$, à l'issue de laquelle le bras ayant la meilleure moyenne empirique est sélectionné jusqu'au temps T. Cette stratégie appelée parfois Explore-Then-Commit peut avoir un regret logarithmique si m est choisi de l'ordre de $\log(T)/\Delta_{\min}^2$ avec $\Delta_{\min} = \min_{a:\Delta_a>0} \Delta_a$. Toutefois, ce choix n'est pas réaliste car il requiert de connaître l'écart entre le meilleur bras et le bras qui en est le plus proche (en plus de l'horizon T). Il est possible de corriger le problème en utilisant une phase d'exploration de durée adaptative, mais sans atteindre un regret optimal, comme discuté dans l'article [9].

Une autre manière de corriger la stratégie gloutonne consiste à ajouter à chaque instant la possibilité de choisir un bras uniformément au hasard. Étant donnée une suite de probabilités d'exploration $\varepsilon = (\varepsilon_t)_{t \in \mathbb{N}^*}$, la stratégie ε-greedy choisit à l'instant t le bras ayant la meilleure moyenne empirique avec probabilité $1 - \varepsilon_t$ et, avec probabilité ε_t, choisit un bras uniformément au hasard dans $[K]$. Cette stratégie est largement utilisée dans le contexte plus général de l'apprentissage par renforcement [21] mais

souffre de plusieurs défauts. Tout d'abord avec une probabilité d'exploration constante, le regret est forcément linéaire. L'article [2] propose une analyse de la stratégie ε-greedy avec une suite décroissante $\varepsilon_t = C/t$ mais la constante C doit être calibrée en connaissant une borne inférieure sur $\Delta_{\min}$ pour obtenir un regret logarithmique.

Nous allons maintenant décrire une famille d'algorithmes ne nécessitant aucune connaissance a priori sur les moyennes des bras (ni sur l'horizon T) pour atteindre un regret logarithmique (quasi) optimal.

Le principe d'optimisme et l'algorithme d'UCB Plutôt que d'utiliser seulement des estimateurs des moyennes des bras, les algorithmes de type UCB sont basés sur des *intervalles de confiance* sur les moyennes. Pour chaque bras a, étant données les observations collectées jusqu'à l'instant t, on peut construire un intervalle $[\mathrm{LCB}_a(t), \mathrm{UCB}_a(t)]$ dans lequel se trouve μ_a avec forte probabilité. La borne inférieure est noté LCB pour Lower Confidence Bound et la borne supérieure UCB pour Upper Confidence Bound. Le principe d'optimisme consiste à considérer que le vrai modèle est le modèle le plus favorable qui reste statistiquement compatible avec les observations effectuées : dans ce modèle optimiste, la moyenne du bras a est égale à la plus grande valeur possible $\mathrm{UCB}_a(t)$ et l'agent choisit alors

$$A_{t+1} = \operatorname*{argmax}_{a \in [K]} \ \mathrm{UCB}_a(t). \tag{2.1}$$

Il s'agit d'un cas particulier de ce qu'on appelle une *politique d'indice* : l'algorithme calcule un indice pour chaque bras a, ici $\mathrm{UCB}_a(t)$, qui ne dépend que de l'historique des récompenses observées pour ce bras.

La formule (2.1) définit une famille d'algorithmes, dont chaque instanciation dépend de l'expression de l'UCB, qui est guidée par les hypothèses qui sont faites sur les distributions des bras. L'algorithme le plus populaire, que nous appellerons UCB(α) utilise l'indice

$$\mathrm{UCB}_a(t) = \widehat{\mu}_a(t) + \sqrt{\frac{\alpha \log(t)}{N_a(t)}} \ \text{ si } \ N_a(t) > 0,$$

et $\mathrm{UCB}_a(t) = +\infty$ si $N_a(t) = 0$, pour un paramètre $\alpha \in \mathbb{R}_+$. Un algorithme de cette forme a d'abord été proposé pour des distribution Gaussiennes de variance σ^2 avec le choix de paramètre $\alpha = 2\sigma^2$ [11]. En 2002, Auer et al. [2] donnent une analyse à temps fini de l'algorithme UCB1 pour des distributions à support borné dans $[0, 1]$, qui revient à choisir le paramètre $\alpha = 2$.

L'indice utilisé par l'algorithme $\mathrm{UCB}_a(t)$ a le bon goût d'être explicite et facile à interpréter : il est la somme d'un terme d'exploitation, la moyenne empirique $\widehat{\mu}_a(t)$ du bras et d'un terme appelé *bonus d'exploration*, $\sqrt{\alpha \log(t)/N_a(t)}$. Le bonus d'exploration est grand pour les bras peu tirés jusqu'alors. Par ailleurs, si un bras n'est pas tiré depuis un certain temps et que $\widehat{\mu}_a(t)$ et $N_a(t)$ ne sont pas mis à jour depuis longtemps, le terme en $\log(t)$ fait tout de même augmenter l'indice de ce bras, ce qui va conduire à sa sélection. Comme on le verra dans l'analyse de l'algorithme UCB que l'on donne ci-dessous, la présence de ce $\log(t)$ vient du fait que les intervalles de confiance sont calibrés pour un niveau de confiance qui augmente avec t. Une illustration d'algorithme UCB est donné sur la figure 2.2, où l'on voit que l'algorithme sélectionne massivement le bras optimal.

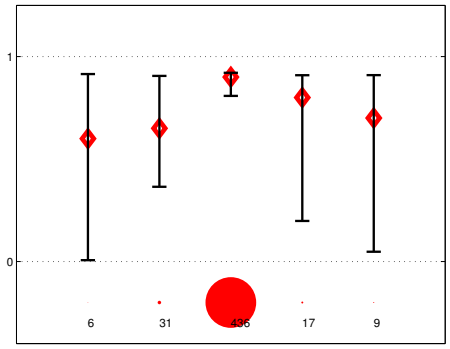

FIGURE 2.2 – *Intervalles de confiance pour les moyennes (en rouge) de chaque bras d'un bandit à 5 bras après $T = 500$ tours de jeu d'un algorithme UCB. Le nombre de tirages de chaque bras est indiqué en dessous.*

Une analyse de l'algorithme UCB(α) Les distributions gaussiennes de variances connues et les distributions à support borné sont des exemples de distributions sous-Gaussiennes.

Définition 2.2.1. *Une distribution de moyenne μ est σ^2 sous-gaussienne si sa fonction génératrice des moments est majorée par celle d'une distribution gausienne de même moyenne et de variance σ^2 :*

$$\forall \lambda \in \mathbb{R},\ \log \mathbb{E}\left[e^{\lambda(X-\mu)}\right] \leqslant \frac{\lambda^2\sigma^2}{2}$$

Les lois gaussiennes de variance σ^2 sont évidemment σ^2 sous-gaussiennes, et d'après le lemme de Hoeffding, une loi à support borné

dans un intervalle $[a, b]$ est $\frac{(b-a)^2}{4}$ sous-gaussienne. L'hypothèse faite sur la fonction génératrice des moments dans la définition ci-dessus permet d'obtenir une inégalité de concentration, ce qui est crucial pour la construction d'un intervalle de confiance.

Lemme 2.2.2 (inégalité de Hoeffding)**.** *Soit (X_i) des réalisations i.i.d. d'un distribution σ^2 sous-gaussienne de moyenne μ. Pour tout $x > 0$,*

$$\mathbb{P}\left(\frac{1}{t}\sum_{s=1}^{t} X_i < \mu - x\right) \leqslant e^{-\frac{tx^2}{2\sigma^2}} \quad et \quad \mathbb{P}\left(\frac{1}{t}\sum_{s=1}^{t} X_i > \mu + x\right) \leqslant e^{-\frac{tx^2}{2\sigma^2}}$$

Nous proposons ci-dessous une analyse simple de l'algorithme UCB(α) pour des distributions σ^2 sous-gaussiennes avec le choix de paramètre $\alpha = 6\sigma^2$. Il s'agit d'une analyse à temps fini dans l'esprit de celle donnée par [2] pour les distributions bornées.

Théorème 2.2.3. *Pour des bras qui sont des distributions sous-gaussiennes de paramètre σ^2, l'algorithme UCB(α) avec le choix de paramètre $\alpha = 6\sigma^2$ vérifie pour tout bras sous-optimal a,*

$$\mathbb{E}[N_a(T)] \leqslant \frac{24\sigma^2 \log(T)}{(\mu_\star - \mu_a)^2} + 1 + \frac{\pi^2}{3}.$$

Démonstration. Pour fixer les idées, on suppose que le bras 1 est optimal et on cherche à majorer le nombre de sélections d'un bras sous optimal a. Dans la décomposition ci-dessous on considère deux cas possibles : soit le bras 1 est bien estimé ($\mathrm{UCB}_1(t) > \mu_1$) (ce qui est censé se produire avec forte probabilité), soit il est mal estimé ($\mathrm{UCB}_1(t) \leqslant \mu_1$) :

$$\begin{aligned} N_a(T) &= \sum_{t=0}^{T-1} \mathbb{1}(A_{t+1} = a) \\ &= 1 + \sum_{t=K}^{T-1} \mathbb{1}(A_{t+1} = a) \cap (\mathrm{UCB}_1(t) \leqslant \mu_1) + \sum_{t=K}^{T-1} \mathbb{1}(A_{t+1} = a) \cap (\mathrm{UCB}_1(t) > \mu_1) \\ &\leqslant 1 + \sum_{t=K}^{T-1} \mathbb{1}(\mathrm{UCB}_1(t) \leqslant \mu_1) + \sum_{t=K}^{T-1} \mathbb{1}(A_{t+1} = a) \cap (\mathrm{UCB}_a(t) \geqslant \mathrm{UCB}_1(t) > \mu_1). \end{aligned}$$

La seconde égalité utilise le fait que l'algorithme UCB tire une fois chaque bras (et donc le bras a) pendant les K premiers instants et l'inégalité utilise la définition de l'algorithme UCB : si le bras a est sélectionné à l'instant $t+1$, l'indice $\mathrm{UCB}_a(t)$ est supérieur à tous les autres, en particulier à $\mathrm{UCB}_1(t)$. En prenant l'espérance, on obtient alors

$$\mathbb{E}_\nu[N_a(T)] \leqslant 1 + \underbrace{\sum_{t=K}^{T-1} \mathbb{P}(\mathrm{UCB}_1(t) \leqslant \mu_1)}_{\textbf{(A)}} + \underbrace{\sum_{t=K}^{T-1} \mathbb{P}(A_{t+1} = a, \mathrm{UCB}_a(t) > \mu_1)}_{\textbf{(B)}}$$

Pour majorer le **terme (A)**, on va utiliser l'inégalité de Hoeffding. Observons qu'on ne peut pas l'appliquer directement car le nombre d'observations du bras 1, $N_1(t)$, est *aléatoire*. Pour gérer ce problème, une solution simple consiste à utiliser une borne de l'union. Avec le choix

$$\begin{aligned} \mathrm{UCB}_b(t) &= \widehat{\mu}_b(t) + \sqrt{\frac{6\sigma^2 \log(t)}{N_b(t)}} \\ \mathrm{LCB}_b(t) &= \widehat{\mu}_b(t) - \sqrt{\frac{6\sigma^2 \log(t)}{N_b(t)}} \end{aligned}$$

pour tout bras $b \in [K]$, on montre que

$$\mathbb{P}\left(\mathrm{UCB}_b(t) < \mu_b\right) \leqslant \frac{1}{t^2} \tag{2.2}$$

$$\text{et} \quad \mathbb{P}\left(\mathrm{LCB}_b(t) > \mu_b\right) \leqslant \frac{1}{t^2}\,. \tag{2.3}$$

En effet, en notant $Y_{b,s}$ la s-ème observation du bras b, on a pour $t > K$

$$\begin{aligned} \mathbb{P}\left(\mathrm{UCB}_b(t) < \mu_b\right) &= \mathbb{P}\left(\widehat{\mu}_b(t) + \sqrt{\frac{6\sigma^2 \log(t)}{N_b(t)}} < \mu_b\right) \\ &= \sum_{s=1}^{t} \mathbb{P}\left(N_b(t) = s, \frac{1}{s}\sum_{i=1}^{s} Y_{b,i} + \sqrt{\frac{6\sigma^2 \log(t)}{s}} < \mu_b\right) \\ &\leqslant \sum_{s=1}^{t} \mathbb{P}\left(\frac{1}{s}\sum_{i=1}^{s} Y_{b,i} < \mu_b - \sqrt{\frac{6\sigma^2 \log(t)}{s}}\right) \\ &\leqslant \sum_{s=1}^{t} \frac{1}{t^3} = \frac{1}{t^2}, \end{aligned}$$

où la seconde inégalité utilise le lemme 2.2.2. La preuve de l'inégalité (2.3) est similaire. En utilisant l'inégalité (2.2), on obtient la majoration

$$\textbf{(A)} \leqslant \sum_{t=K}^{T-1} \frac{1}{t^2} \leqslant \sum_{t=1}^{\infty} \frac{1}{t^2} = \frac{\pi^2}{6}.$$

Pour majorer le **terme (B)**, on distingue deux cas en fonction de la position de μ_a par rapport à la borne inférieure de l'intervalle de confiance :

$$\begin{aligned} (B) &\leqslant \sum_{t=K}^{T-1} \mathbb{P}\left(A_{t+1} = a, \mathrm{UCB}_a(t) > \mu_1, \mathrm{LCB}_a(t) \leqslant \mu_a\right) + \sum_{t=K}^{T-1} \mathbb{P}\left(\mathrm{LCB}_a(t) > \mu_a\right) \\ &\leqslant \sum_{t=K}^{T-1} \mathbb{P}\left(A_{t+1} = a, \mathrm{UCB}_a(t) > \mu_1, \mathrm{LCB}_a(t) \leqslant \mu_a\right) + \frac{\pi^2}{6}, \end{aligned}$$

où la deuxième inégalité utilise la majoration (2.3). On observe maintenant que si $UCB_a(t) > \mu_1$ et $\mathrm{LCB}_a(t) \leqslant \mu_a$, avec $\mu_1 > \mu_a$ il vient que μ_1 et μ_a appartiennent à l'intervalle de confiance $[\mathrm{LCB}_a(t); \mathrm{UCB}_a(t)]$, en particulier l'écart $\mu_1 - \mu_a$ est inférieur à la largeur de cet intervalle de confiance :

$$\mu_1 - \mu_a \leqslant 2 \times \sqrt{\frac{6\sigma^2 \log(t)}{N_a(t)}} \quad \Rightarrow \quad N_a(t) \leqslant \frac{24\sigma^2 \log(T)}{(\mu_1 - \mu_a)^2}$$

Ainsi

$$\begin{aligned} \textbf{(B)} \quad &\leqslant \quad \sum_{t=K}^{T-1} \mathbb{P}\left(A_{t+1} = a, N_a(t) \leqslant \frac{24\sigma^2 \log(T)}{(\mu_1 - \mu_a)^2} \right) + \frac{\pi^2}{6} \\ &\leqslant \quad \frac{24\sigma^2 \log(T)}{(\mu_1 - \mu_a)^2} + \frac{\pi^2}{6}. \end{aligned}$$

On conclut en sommant les majorations obtenues pour **(A)** et **(B)**. ■

De l'optimalité d'UCB L'article [4] présente une analyse plus sophistiquée de l'algorithme UCB(α) pour les distributions sous-gaussiennes où il est montré que pour le choix $\alpha = 2\sigma^2$ l'algorithme vérifie

$$\mathbb{E}[N_a(T)] \leqslant \frac{2\sigma^2}{(\mu_\star - \mu_a)^2} \log(T) + o\left(\log(T)\right)$$

pour tout bras sous-optimal a. Une borne inférieure, prouvée par Lai et Robbins en 1985 [13] montre que pour des distributions gaussiennes, ce nombre de tirages des bras sous-optimaux (et donc le regret associé) est *minimal*, au moins dans un régime asymptotique. Le résultat de Lai et Robbins s'applique à des modèles de bandits dont les bras appartiennent à une famille de distributions paramétriques simple (Gaussiennes de variance connue, Bernoulli, Poisson ou plus généralemenet une famille exponentielle à un paramètre). Il stipule que tout algorithme uniformément efficace [2] vérifie, pour tout bras sous optimal a,

$$\liminf_{T \to \infty} \frac{\mathbb{E}[N_a(T)]}{\log(T)} \geqslant \frac{1}{\mathrm{KL}(\nu_a, \nu_{a_\star})},$$

où $\mathrm{KL}(\nu_a, \nu_{a_\star})$ est la divergence de Kullback-Leibler entre ν_a et la distribution du bras optimal $\nu_{a_\star}$. Dans le cas Gaussien évoqué ci-dessus, on a $\mathrm{KL}(\nu_a, \nu_{a_\star}) = \frac{(\mu_\star - \mu_a)^2}{2\sigma^2}$, ce qui justifie l'optimalité annoncée.

2. un algorithme est dit uniformément efficace si $\mathcal{R}_\nu(\mathcal{A}, T) = o(T^\alpha)$ pour tout $\alpha \in (0, 1)$ et pour tout modèle de bandit ν dans la classe de distributions considérée

Pour les distributions de Bernoulli, l'expression de la borne inférieure est moins explicite car on a

$$\mathrm{KL}(\nu_a, \nu_{a_\star}) = \mu_a \log\left(\frac{\mu_a}{\mu_\star}\right) + (1-\mu_a)\log\left(\frac{1-\mu_a}{1-\mu_\star}\right).$$

Les distributions de Bernoulli étant à support dans $[0,1]$, on peut appliquer l'algorithme UCB(α) avec le paramètre $\alpha = 1/2$, qui vérifie

$$\mathbb{E}[N_a(T)] \leqslant \frac{1}{2(\mu_\star - \mu_a)^2}\log(T) + o\left(\log(T)\right).$$

Or d'après l'inégalité de Pinsker, $\mathrm{KL}(\nu_a, \nu_{a_\star}) > 2(\mu_\star - \mu_a)^2$, et l'algorithme UCB(1/2) n'atteint donc pas la borne inférieure dans ce cas.

Il est possible de construire des algorithmes dont le regret atteint la borne inférieure de Lai et Robbins pour le cas des distributions de Bernoulli. On peut même proposer un algorithme UCB mais dont l'indice $\mathrm{UCB}_a(t)$ est moins explicite que celui donné ci-dessus (il s'exprime en fonction de la divergence de Kullback-Leibler intervenant dans la borne inférieure) : cet algorithme est appelé kl-UCB [4]. Toutefois, l'algorithme UCB(1/2) – appelé parfois simplement UCB – offre un bon compromis entre simplicité et robustesse : il atteint un regret logarithmique pour n'importe quelle distribution de récompense bornée, et ce regret est « quasi-optimal » pour le cas particulier des distributions de Bernoulli.

2.2.2 Un algorithme pour trouver le meilleur bras

Des stratégies d'échantillonage adaptatif de distributions peuvent être proposées pour des objectifs différents de celui de maximiser la somme des observations obtenues. Nous décrivons dans cette partie un algorithme pour l'identification du meilleur bras, également basé sur l'utilisation d'intervalles de confiance. Une stratégie pour l'identification du meilleur bras est constituée de trois composantes : une règle d'échantillonage A_t donnant le bras choisi à l'instant t, une règle d'arrêt τ et une règle de recommandation $\widehat{a}_\tau$ qui propose un candidat pour le bras $a_\star$.

Plusieurs formalisations mathématiques du problème ont été considérées et on s'intéresse ici à l'identification du meilleur bras à niveau de confiance fixé [7]. L'objectif est de garantir que $\mathbb{P}(\widehat{a}_\tau = a_\star) \geqslant 1-\delta$, tout en minimisant le nombre total d'échantillons utilisés τ (en espérance ou en forte probabilité, car c'est en général une variable aléatoire) pour arriver à cette recommandation. Une autre partie de la littérature s'intéresse à l'identification à budget fixé où $\tau = n$ et on cherche à minimiser la probabilité d'erreur sur le bras recommandé après avoir observé n échantillons [1].

L'algorithme LUCB Pour des récompenses bornées dans [0,1] l'article [10] propose l'algorithme LUCB, basé lui aussi sur des intervalles de confiance $\mathcal{I}_a(t) = [\mathrm{LCB}_a(t), \mathrm{UCB}_a(t)]$ pour chaque bras a, mais exploitant cette fois la borne supérieure et la borne inférieure. L'algorithme LUCB est même proposé pour l'objectif de trouver les m meilleurs bras, mais nous ne le présentons que pour le cas particulier $m = 1$.

À l'instant $t + 1$, l'algorithme LUCB sélectionne deux bras :

— le meilleur bras empirique $B_t = \mathrm{argmax}_{a \in [K]} \widehat{\mu}_a(t)$;

— le challenger qui a l'UCB le plus grand :

$$C_t = \mathrm{argmax}_{c \in [K], c \neq B_t} \mathrm{UCB}_c(t)$$

L'algorithme arrête son échantillonage lorsque la borne de confiance inférieure du candidat meilleur bras est supérieure à la borne de confiance supérieure du challenger C_t (et donc de tous les autres bras) :

$$\tau = \inf \{t \in \mathbb{N} : \mathrm{LCB}_{B_t}(t) > \mathrm{UCB}_{C_t}(t)\}$$

L'illustration de cette condition de terminaison est donnée sur la figure 2.3, où l'on voit bien la séparation entre l'intervalle de confiance du meilleur bras et ceux des autres bras. Lorsque l'algorithme termine, le bras recommandé est le meilleur bras empirique $\widehat{a}_\tau = B_\tau$.

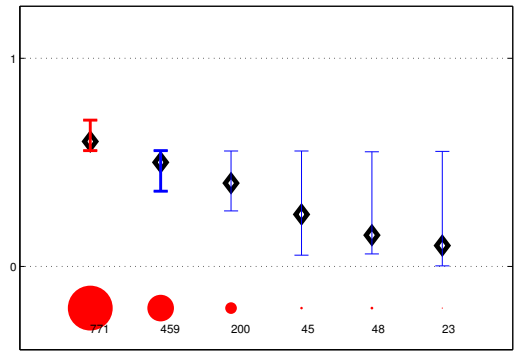

FIGURE 2.3 – *Intervalles de confiance et nombre de tirages des différents bras au moment où l'algorithme LUCB termine.*

La forme des intervalles de confiance utilisés par LUCB est essentiellement la même que celle utilisée par UCB pour des récompenses bornées, mais leur calibration est différente. En effet, un algorithme pour l'identification du meilleur bras dépend d'un niveau de risque maximal δ, et

les intervalles de confiance vont donc dépendre de ce paramètre. Plus précisément, ils sont choisis de sorte que l'événement

$$\mathcal{G} = \Big(\forall t \in \mathbb{N}^*, \forall a \in [K], \mu_a \in [\mathrm{LCB}_a(t), \mathrm{UCB}_a(t)]\Big)$$

soit de probabilité $1-\delta$. Sur cet événement, il n'est pas difficile de se convaincre que l'algorithme ne peut pas s'arrêter et recommander un bras sous-optimal a, puisqu'alors on aurait $\mu_a \geqslant \mathrm{LCB}_a(\tau) > \mathrm{UCB}_{a_\star}(\tau) \geqslant \mu_{a_\star}$, ce qui contredit le fait que $\mu_a < \mu_{a_\star}$.

Un choix valide des bornes de confiance est

$$\mathrm{UCB}_a(t) = \widehat{\mu}_a(t) + \sqrt{\frac{\log\left(\frac{\pi^2 t^2}{3\delta}\right)}{2N_a(t)}} \quad \text{et} \quad \mathrm{LCB}_a(t) = \widehat{\mu}_a(t) - \sqrt{\frac{\log\left(\frac{\pi^2 t^2}{3\delta}\right)}{2N_a(t)}}. \tag{2.4}$$

En effet, on peut majorer la probabilité de l'événement complémentaire

$$\mathcal{G}^c \subseteq \bigcup_{a\in[K]} \bigcup_{s\in\mathbb{N}^*} \left(\widehat{\mu}_{a,s} + \sqrt{\frac{\log\left(\frac{\pi^2 s^2}{3\delta}\right)}{2s}} < \mu_a\right) \bigcup \left(\widehat{\mu}_{a,s} - \sqrt{\frac{\log\left(\frac{\pi^2 s^2}{3\delta}\right)}{2s}} > \mu_a\right)$$

en utilisant l'inégalité de Hoeffding pour chaque valeur de a et de s puis en sommant :

$$\mathbb{P}\left(\mathcal{G}^c\right) \leqslant \sum_{a=1}^{K} \sum_{s=1}^{\infty} \left[\mathbb{P}\left(\widehat{\mu}_{a,s} < \mu_a - \sqrt{\frac{\log\left(\frac{\pi^2 s^2}{3\delta}\right)}{2s}}\right) + \mathbb{P}\left(\widehat{\mu}_{a,s} > \mu_a + \sqrt{\frac{\log\left(\frac{\pi^2 s^2}{3\delta}\right)}{2s}}\right)\right] \leqslant \delta.$$

L'algorithme LUCB utilisant les bornes (2.4) et donc garanti de trouver le meilleur bras avec probabilité plus grande que $1-\delta$. L'analyse donnée par [10] permet de plus de prouver que le nombre de rounds utilisés par l'algorithme satisfait

$$\mathbb{E}[\tau] = O\left(\left[\sum_{a=1}^{K} \frac{1}{\Delta_a^2}\right] \log\left(\frac{1}{\delta}\right)\right)$$

où $\Delta_a = \mu_\star - \mu_a$ si $\mu_a < \mu_\star$ et $\Delta_{a_\star} = \min_{a\neq a_\star} \Delta_a$. Ce résultat s'interprète en disant que chaque bras doit être échantillonné proportionnellement à l'inverse de son gap au carré Δ_a^2. Des travaux ultérieurs se sont intéressés au nombre *minimal* d'échantillons nécessaires pour identifier le meilleur bras avec probabilité plus grande que $1-\delta$ en proposant une borne inférieure sur $\mathbb{E}[\tau]$ et des algorithmes atteignant cette borne [8], mais l'expression de la borne inférieure est moins explicite que dans le cas du regret.

UCB pour trouver le meilleur bras ? En comparant la figure 2.2 et la figure 2.3 on voit que UCB et LUCB échantillonnent les bras de manière assez différente. Alors qu'UCB se focalise sur le meilleur bras, LUCB va explorer aussi de manière significative les deuxième et troisième meilleurs bras. LUCB n'étant pas pénalisé par le tirage de bras sous-optimaux, cet algorithme va réaliser une bonne estimation des moyennes des bras sous-optimaux, proche du meilleur bras, afin de pouvoir recommander le bras qui a la meilleure moyenne empirique lorsqu'il s'arrête. L'algorithme UCB, au contraire, va tirer chaque bras sous-optimal de l'ordre de $\log(T)$ fois et le bras optimal de l'ordre de T fois, donc seule l'estimation obtenue pour la moyenne du bras optimal sera beaucoup plus fiable que celles des moyennes des autres bras.

Une idée naturelle est donc de proposer comme candidat pour le meilleur bras non pas le bras ayant la meilleure moyenne empirique mais *le bras qui a été le plus sélectionné jusqu'alors par l'algorithme UCB*. Cette stratégie est analysée dans l'article [3] dans un régime où le budget d'échantillonage est fixé. Lorsque les récompenses sont bornées, il est montré que le bras $B_T = \text{argmax}_a N_a(T)$ qui a été le plus sélectionné par l'algorithme UCB(α) à l'instant T satisfait, pour $\alpha > 1$

$$\mathbb{E}\left[\mu_\star - \mu_{B_T}\right] = O\left(\frac{1}{T^{\alpha-1}}\right).$$

Ainsi en prenant un paramètre d'exploration α un peu plus grand que la valeur recommandée plus haut $\alpha = 1/2$ pour la minimisation du regret, on peut avoir des garanties pour le meilleur bras trouvé par UCB.

Ces garanties sont loin d'être optimales, car on peut trouver des algorithmes pour lesquels le regret simple $\mathbb{E}\left[\mu_\star - \mu_{B_T}\right]$ décroît exponentiellement en T [1] et les auteurs de [3] montrent qu'un algorithme minimisant le regret (cumulé) ne peut pas être optimal pour minimiser le regret simple (dans un régime où l'horizon T est grand). Ainsi les solutions *optimales* de la maximisation des récompenses et de l'identification du meilleur bras dans un modèle de bandit sont très différentes, mais utiliser UCB et recommander le bras le plus tiré constitue une heuristique tout à fait raisonnable pour trouver le meilleur bras. Elle est notamment utilisée dans l'algorithme de Monte Carlo Tree Search que nous présentons dans la partie suivante.

2.3 Des bandits au Monte Carlo Tree Search

La partie précédente a décrit la brique de base : l'algorithme UCB pour le problème des bandits. Dans cette partie, nous présentons un algorithme de Monte Carlo Tree Search (MCTS) basé sur UCB et appelé UCT. Ceci se fera en plusieurs mouvements : le cadre naturel de cet algorithme est celui des processus de décisions Markoviens introduits dans la partie 2.3.1. Nous introduisons une notion centrale pour l'apprentissage par renforcement, les Q-valeurs dans la partie 2.3.2. Pour se mettre en jambe, on commence par le cadre dit « dynamique », pour lequel le MDP est connu, dans la partie 2.3.3. Dans le cadre général, le MDP n'est pas connu, on peut seulement le simuler. Nous présentons l'algorithme MCTS dans ce cadre dans la partie 2.3.4, en faisant l'hypothèse supplémentaire que le nombre d'états est fini. Nous expliquons ensuite comment étendre l'algorithme aux jeux à somme nulle dans la partie 2.3.5, et aux ensembles d'états infinis dans la partie 2.3.6.

2.3.1 Processus de décisions Markoviens

Modèle Comme expliqué plus haut, le modèle le plus courant pour l'apprentissage par renforcement est celui de processus de décision Markovien. En anglais on dit « Markov decision process », ce qui donne l'acronyme MDP que nous utiliserons ici. Dans un MDP, un agent agit dans un environnement stochastique : il s'agit donc de jeux à un joueur, comme pour le modèle des bandits décrit dans la partie précédente. La différence essentielle avec le modèle des bandits est la notion d'état : à chaque instant, l'environnement est dans un état qui est une description complète de la situation courante du système. Notons S l'ensemble d'états. Les K bras du modèle précédent correspondent ici à un ensemble A d'actions, qui sont les choix de l'agent. Pour un ensemble X, notons $\mathcal{D}(X)$ l'ensemble des distributions sur X. La dynamique du MDP est donnée par la fonction de transition $\delta : S \times A \to \mathcal{D}(\mathbb{R} \times S)$, que l'on interprète comme suit : $\delta(s, a)$ est la distribution obtenue en choisissant l'action a depuis l'état s. Ainsi, $\delta(s, a)(r, s')$ est la probabilité de transitionner vers l'état s' et d'obtenir une récompense immédiate de r.

Une trajectoire est une suite $(s_0, a_0, r_0, s_1, a_1, r_1, \dots)$, décrivant une évolution possible du système : pour tout $i \in \mathbb{N}$, depuis l'état s_i, l'action a_i est jouée, et cela mène vers l'état s_{i+1} avec récompense r_i. La récompense totale d'une trajectoire est la somme des récompenses immédiates : $r_0 + r_1 + \dots$. Dans tous les cas pratiques, les trajectoires sont finies et donc la récompense totale est bien définie. Mais si l'on considère des trajectoires infinies, il se peut que cette somme n'existe pas. La solution classique

est d'introduire un taux d'escompte : il s'agit d'une constante $\gamma \in (0,1)$ qui sert à réduire l'influence des récompenses obtenues dans le futur. La récompense totale escomptée est

$$\sum_{i \geqslant 0} \gamma^i r_i = r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$$

Mathématiquement, à condition que les récompenses soient bornées, cette somme infinie converge et permet donc de se placer sur des bases mathématiques solides pour étudier les MDPs. Il y a une autre raison d'introduire le taux d'escompte : il formalise l'idée importante qu'une récompense obtenue maintenant est plus importante que la même récompense obtenue plus tard. Ainsi, même dans le cas où les trajectoires sont finies il est courant d'utiliser un taux d'escompte.

Stratégies L'agent est en charge du choix des actions : il s'appuie pour cela sur une stratégie. Les stratégies les plus simples, dites pures et positionnelles, sont des fonctions $\sigma : S \to A$: à chaque état la stratégie désigne une action à jouer. On peut introduire deux mécanismes pour enrichir ces stratégies : l'aléatoire pour permettre à la stratégie de décrire une distribution sur les actions et la mémoire pour retenir un certain nombre d'informations sur la trajectoire obtenue jusque là. Les stratégies sont souvent appelées politiques, c'est le même objet.

Ayant fixé une stratégie σ, on définit une mesure de probabilité $\mathbb{P}_\sigma$ sur l'ensemble des trajectoires infinies. Mathématiquement, elle est induite par la définition suivante et le théorème d'extension unique de mesure de Carathéodory : la trajectoire finie $\pi = (s_0, a_0, r_0, s_1, \dots, s_k)$ a probabilité

$$\mathbb{P}_\sigma(\pi) = \prod_{i=0}^{k-1} \delta(s_i, a_i)(r_i, s_{i+1}).$$

Pour chaque $i \in \mathbb{N}$, on définit la variable aléatoire R_i décrivant la récompense immédiate obtenue à l'étape i. La récompense totale escomptée est une variable aléatoire G définie par $G = \sum_{i \geqslant 0} \gamma^i R_i$. Ceci permet de quantifier numériquement la qualité d'une stratégie, que l'on appelle la valeur :

$$\mathrm{val}_\sigma(s) = \mathbb{E}_s^\sigma[G] = \mathbb{E}_s^\sigma\left[\sum_{i \geqslant 0} \gamma^i R_i\right],$$

où l'espérance porte sur les trajectoires commençant en s. La définition ci-dessus induit une fonction de valeur $\mathrm{val}_\sigma : S \to \mathbb{R}$.

Objectif On dit qu'une stratégie σ est optimale si pour toute stratégie σ', on a $\text{val}_\sigma \geqslant \text{val}_{\sigma'}$. Il n'est pas évident qu'il existe une stratégie optimale : en effet, pour deux états s_0 et s_1, il pourrait exister deux stratégies σ_0 et σ_1 telle que σ_0 soit meilleure depuis s_0 et σ_1 depuis s_1. Heureusement, il n'en est rien : on peut montrer qu'il existe une stratégie optimale qui, de plus, est pure et positionnelle [16]. On note σ_* une stratégie optimale (il peut en exister plusieurs), et val_* la fonction de valeur de σ_* (qui ne dépend donc pas du choix de σ_*).

L'objectif algorithmique principal et idéalisé de l'apprentissage par récompense est le suivant : calculer une stratégie optimale pour un MDP donné. Que signifie « un MDP donné » ? On distingue deux situations :

— le MDP est petit et connu : S est un ensemble d'états fini et la fonction $\delta : S \times A \to \mathcal{D}(\mathbb{R} \times S)$ est donnée en entrée, en particulier elle tient dans la mémoire d'un ordinateur. On parle du cadre « dynamique » parce qu'il existe des algorithmes très efficaces dans ce cadre basés sur la programmation dynamique ;

— l'autre cas, le plus courant et le plus utile, est celui où l'ensemble d'états S est soit fini mais très grand, soit infini. On suppose dans ce cas là que le MDP peut seulement être simulé, comme une boîte noire : depuis un état s l'agent choisit une action a et le système retourne une description du nouvel état s' ainsi que la récompense immédiate r obtenue en suivant la distribution $\delta(s, a)$.

Le second cadre est celui qui nous intéresse ici et pour lequel les algorithmes MCTS ont été développés. En effet, le nombre de configurations possibles dans le jeu de Go est fini, mais gigantesque : il est impossible de représenter une fonction définie sur toutes les configurations du jeu de Go.

2.3.2 Les Q-valeurs

La notion de Q-valeur est très proche de celle de valeur que nous avons défini plus haut, mais elle est beaucoup plus pratique pour la construction d'algorithmes. Pour une stratégie σ, on définit les Q-valeurs de σ par l'équation suivante :

$$Q_\sigma(s, a) = \mathbb{E}^\sigma_{s,a}[G],$$

où l'espérance porte sur les trajectoires commençant en s, a : on part de l'état s et la première action choisie est a. Ainsi Q_σ est une fonction $Q_\sigma : S \times A \to \mathbb{R}$. Plus généralement, une fonction $Q : S \times A \to \mathbb{R}$ est appelée une Q-fonction. La remarque importante est qu'une Q-fonction induit une stratégie :

$$\sigma(s) = \text{argmax } \{Q(s, a) : a \in A\}.$$

Mais une Q-fonction apporte plus d'informations : elle quantifie pour chaque paire d'un état et d'une action, l'intérêt de choisir cette action depuis cet état. Pour cette raison, au lieu de manipuler des stratégies, la plupart des algorithmes d'apprentissage par renforcement et tous ceux que nous décrivons ici, manipulent des Q-fonctions.

On note Q_* les Q-valeurs d'une stratégie optimale : comme pour le cas des valeurs, ceci ne dépend pas de la stratégie optimale choisie.

2.3.3 L'algorithme d'amélioration de stratégies

Avant de présenter l'algorithme de Monte Carlo Tree Search, préparons le terrain en introduisant l'algorithme d'amélioration de stratégies, appelé en anglais « policy iteration ». Cet algorithme est en fait le dénominateur commun de nombreux algorithmes en apprentissage par renforcement. Son principe est de construire une suite de stratégies $\sigma_0, \sigma_1, \sigma_2, \ldots$, chacune améliorant la précédente.

Pour cela, on introduit deux objectifs :

— **Évaluation** : étant donné une stratégie σ, comment calculer ses Q-valeurs ?

— **Amélioration** : étant donné les Q-valeurs d'une stratégie σ, comment construire une stratégie avec des Q-valeurs plus grandes ?

Rappelons que dans le cadre dynamique, le MDP est petit et connu : S est un ensemble d'états fini et la fonction $\delta : S \times A \to \mathcal{D}(\mathbb{R} \times S)$ est donnée en entrée.

Évaluation Pour calculer les Q-valeurs d'une stratégie σ, on s'appuie sur les équations de Bellman qui, ici, prennent la forme suivante :

$$Q_\sigma(s,a) = \sum_{(r,s') \in \delta(s,a)} \delta(s,a)(r,s')(r + \gamma \cdot Q_\sigma(s', \sigma(s'))).$$

Cette équation dit simplement que, pour évaluer σ depuis (s,a), il suffit d'évaluer une étape, à savoir (s,a,r,s'), puis de continuer de manière récursive. Il y a plusieurs manières d'utiliser ces équations, la plus efficace s'appuie sur le fait que Q_σ est en fait l'unique solution des équations ci-dessus, et plus précisément le plus petit point fixe de l'opérateur sous-jacent. Ainsi, Q_σ peut être approximé efficacement à l'aide d'un algorithme d'itérations de valeurs.

Amélioration Une fois Q_σ connu, il est naturel de définir σ' comme ci-dessus :

$$\sigma'(s) = \text{argmax } \{Q_\sigma(s,a) : a \in A\}.$$

On peut montrer (et c'est un théorème fondamental) que σ' améliore σ, au sens $\text{val}_{\sigma'} \geqslant \text{val}_{\sigma}$. Mieux, l'inégalité est stricte si et seulement si σ n'est pas optimale. Ainsi, l'algorithme d'amélioration de stratégies s'arrête quand il n'est plus possible d'améliorer et on obtient alors la stratégie optimale.

2.3.4 L'algorithme de Monte Carlo Tree Search

On se place maintenant dans le cadre général de l'apprentissage par renforcement où le MDP peut seulement être simulé. Remarquons que c'est une généralisation du problème des bandits présenté plus haut : c'est seulement en simulant le système par le choix d'actions que l'agent acquiert des informations sur le système. Il s'agit donc de décrire des techniques statistiques basées sur l'échantillonnage. Le même compromis apparaît : exploration *versus* exploitation. D'un côté, on souhaite choisir les meilleures actions afin d'optimiser la récompense totale, c'est l'exploitation. Mais de l'autre, on doit également explorer le système, afin de potentiellement découvrir de meilleurs coups que ceux découverts jusque là. L'idée de l'algorithme MCTS, et plus précisément de la variante UCT, est de s'appuyer sur l'algorithme UCB présenté plus haut pour résoudre ce compromis.

Dans la suite nous supposons que, pour tous les MDPs que nous considérons, les trajectoires sont finies. Dit autrement, le MDP contient un état puits où les trajectoires s'arrêtent et qui est atteint presque sûrement quels que soient les choix de l'agent. C'est le cas de la plupart des jeux : après un nombre fini de coups, soit un joueur est déclaré vainqueur, soit la partie est déclaré nulle.

Dans un premier temps, nous allons considérer le cas d'un ensemble d'états fini mais grand, et discuterons dans la partie 2.3.6 du cadre plus général où cet ensemble est infini. Comme pour l'algorithme précédent, nous manipulons une Q-fonction $Q : S \times A \to \mathbb{R}$. Cependant, comme S est très grand, nous ne voulons pas définir Q sur $S \times A$ tout entier : Q est une fonction partielle. Intuitivement, on ne va définir Q que sur des états jugés intéressants, et être parcimonieux dans le choix des états intéressants pour ne pas utiliser trop d'espace mémoire.

Description haut niveau Commençons par décrire l'algorithme à haut niveau. Comme dans tout algorithme d'apprentissage, il y a une phase d'entraînement, où l'algorithme affine ses paramètres selon les données du problèmes, puis une phase de jeu. Nous allons donc décrire deux procédures : une d'entraînement et une pour jouer, qui prend en entrée l'état courant et choisit une action. De manière assez naïve, l'entraînement est basé sur l'échantillonnage de trajectoires par la stratégie σ induite par la fonction Q :

ce n'est pas aussi simple, mais c'est une première approximation. On pourrait alors encore plus naïvement décrire la fonction de jeu ainsi : l'action est choisie par la stratégie σ, rappelons que $\sigma(s) = \text{argmax } \{Q(s,a) : a \in A\}$. Ce serait assez peu efficace : en effet, même avec un très long entraînement, il se peut que l'on visite des états sur lesquels on a peu d'informations. Pour résoudre ce problème, on va au moment de jouer depuis s procéder également à de l'échantillonnage de trajectoires depuis l'état s pendant un budget de temps fixé (par exemple 10s), afin d'améliorer notre compréhension de la situation depuis cet état et ensuite jouer l'action ayant les meilleurs résultats sur ces échantillons.

Nous arrivons donc à une description un peu plus précise de l'algorithme : entraînement et jeu s'appuient sur une procédure d'échantillonage, que nous décrivons maintenant. Un point de terminologie ici : c'est cette procédure d'échantillonage que l'on appelle MCTS, et dont l'objectif est de choisir la prochaine action à jouer.

Le cadre dynamique, et en particulier l'algorithme d'amélioration de stratégies, sert de guide. En particulier, on étudie les deux problèmes : évaluation d'une stratégie et amélioration. Il y aurait plus à dire sur les liens entre l'algorithme que nous décrivons là et l'algorithme d'amélioration de stratégies.

Évaluation Pour évaluer σ on va simuler un certain nombre de trajectoires avec cette stratégie. Il y a deux problèmes : le premier est que cette approche n'inclut pas d'exploration, et le second est que la stratégie σ n'est pas définie partout, donc il n'est pas clair comment simuler des trajectoires complètes.

La solution proposée au premier problème est de s'appuyer sur l'algorithme UCB. On définit donc :

$$\sigma(s) = \text{argmax } \{Q(s,a) + c(s,a) : a \in A\},$$

où $c(s,a) = \sqrt{\frac{\alpha \log(t)}{N_{s,a}}}$ avec t le nombre de simulations effectuées jusque là, $N_{s,a}$ le nombre de fois que l'action a a été choisie depuis l'état s, et α un paramètre réel positif. C'est l'indice UCB vu plus haut. Notons que pour que $\sigma(s)$ soit bien définie, il faut que chaque action ait été choisie au moins une fois.

Pour le deuxième problème, on distingue trois phases qui se succèdent :

— **Sélection** : tant que la stratégie σ est bien définie, on joue selon cette stratégie. En d'autres termes, pour un sommet s, si toutes les actions a ont été choisies au moins une fois, alors $\sigma(s)$ est bien définie et on l'utilise pour sélectionner une action ;

— **Expansion** : la phase de sélection s'achève la première fois que σ n'est pas définie depuis un sommet s, il existe au moins une action qui n'a jamais été choisie. La phase d'expansion consiste à choisir une telle action. On appelle ceci « expansion » parce qu'on étend la fonction Q en définissant $Q(s,a)$;
— **Playout** : après la phase d'expansion commence la dernière phase, dite de « playout ». À partir de là, la stratégie σ n'est plus définie. La solution la plus simple est de jouer aléatoirement, ou de suivre une heuristique.

Amélioration Chaque simulation, en suivant les trois phases décrites ci-dessus, aboutit à une récompense totale. On utilise cette information pour améliorer la fonction Q. On dit que l'on propage l'information vers le haut (« backpropagation ») : on remonte la trajectoire jusqu'à l'état initial, et on met à jour les valeurs de Q. C'est ici que la distinction entre les trois phases est importante : on ne met à jour les valeurs que pour les paires (s,a) visitées pendant les phases de sélection et d'expansion. En effet, pendant la phase de playout, on joue de manière aléatoire, on visite donc des états potentiellement peu intéressants, et l'on ne veut pas augmenter d'espace mémoire pour ceux-là. Il existe plusieurs possibilités pour mettre à jour les valeurs ; une des plus populaires est dite des différences temporelles (« temporal difference »). La voici en équation, pour mettre à jour en remontant une transition (s,a,r,s',a') :

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a)),$$

où $\alpha \in (0,1)$ est une constante appelée pas. L'idée est de mettre à jour $Q(s,a)$ vers $r + \gamma Q(s',a')$ qui une estimation de la récompense totale obtenue depuis (s,a).

2.3.5 Extension aux jeux à deux joueurs à somme nulle

L'algorithme MCTS a été présenté plus haut dans le cadre des MDPs, donc avec un seul agent. Il s'étend de manière très naturelle aux jeux à deux joueurs à somme nulle. Dans ces jeux, deux joueurs s'affrontent et obtiennent chacun une récompense totale opposée. Autrement dit, si le premier joueur obtient R, alors le deuxième joueur obtient $-R$. C'est le cas des jeux où le résultat est soit gagné (que l'on peut décrire comme récompense 1), soit perdu (récompense -1), soit match nul (récompense 0). Les deux joueurs sont appelés maximiseur et minimiseur, reflétant leurs rôles.

L'algorithme précédent s'étend de manière très naturelle. En effet, la Q-fonction Q induit les stratégies

$$\sigma_{\max}(s) = \text{argmax } \{Q(s,a) + c(s,a) : a \in A\},$$
$$\sigma_{\min}(s) = \text{argmin } \{Q(s,a) - c(s,a) : a \in A\}.$$

On simule donc des trajectoires en s'appuyant sur les deux stratégies à la fois, $\sigma_{\max}$ pour les coups de maximiseur, et $\sigma_{\min}$ pour les coups de minimiseur. En d'autres termes, l'algorithme joue contre lui-même, puisque les deux stratégies sont issues de la Q-fonction Q. Cette idée naturelle est appelée « self-play » : d'un point de vue mathématique, il est loin d'être évident que cette approche permette effectivement de converger vers les valeurs Q_* optimales.

2.3.6 Mettre à profit la puissance des réseaux de neurones

Nous avons, pour simplifier la présentation, représenté la Q-fonction Q de manière explicite en disant qu'elle est définie sur un sous-ensemble de $Q \times A$. Ceci peut être problématique si S est infini, mais également et surtout pour obtenir une stratégie bien définie depuis des états qu'elle n'a jamais vus. C'est la force des réseaux de neurones : même s'ils n'ont jamais vu une situation, ils peuvent extrapoler à partir de situations connues pour prendre une décision pertinente, on dit qu'ils « généralisent ». Pour obtenir un algorithme de MCTS performant, on représente la Q-fonction Q à l'aide d'un réseau de neurones : par exemple, l'entrée est un encodage des états sur un vecteur de dimension fixée et la sortie donne pour chaque action une valeur numérique. L'algorithme est essentiellement le même ; la seule différence réside dans la mise à jour des valeurs, qui doit être adaptée à la représentation choisie.

Bibliographie

[1] J.-Y. AUDIBERT, S. BUBECK et R. MUNOS : Best Arm Identification in Multi-armed Bandits. *Actes de Conference on Learning Theory (COLT)*, 2010.

[2] P. AUER, N. CESA-BIANCHI et P. FISCHER : Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, 2002.

[3] S. BUBECK, R. MUNOS et G. STOLTZ : Pure Exploration in Finitely Armed and Continuous Armed Bandits. *Theoretical Computer Science 412, 1832-1852*, 412:1832–1852, 2011.

[4] O. CAPPÉ, A. GARIVIER, O.-A. MAILLARD, R. MUNOS et G. STOLTZ : Kullback-Leibler upper confidence bounds for optimal sequential allocation. *Annals of Statistics*, 41(3):1516–1541, 2013.

[5] R. COULOM : Efficient selectivity and backup operators in Monte-Carlo Tree Search. *Actes de International Conference Computers and Games, CG*, pp. 72–83, 2006.

[6] R. DEGENNE, W. M. KOOLEN et P. MÉNARD : Non-asymptotic pure exploration by solving games. *Actes de Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[7] E. EVEN-DAR, S. MANNOR et Y. MANSOUR : Action Elimination and Stopping Conditions for the Multi-Armed Bandit and Reinforcement Learning Problems. *Journal of Machine Learning Research*, 7:1079–1105, 2006.

[8] A. GARIVIER et E. KAUFMANN : Optimal best arm identification with fixed confidence. *Actes de Conference on Learning Theory (COLT)*, 2016.

[9] A. GARIVIER, E. KAUFMANN et T. LATTIMORE : On explore-then-commit strategies. *Actes de Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[10] S. KALYANAKRISHNAN, A. TEWARI, P. AUER et P. STONE : PAC subset selection in stochastic multi-armed bandits. *Actes de International Conference on Machine Learning (ICML)*, 2012.

[11] M. KATEHAKIS et H. ROBBINS : Sequential choice from several populations. *Proceedings of the National Academy of Science*, 92:8584–8585, 1995.

[12] L. KOCSIS et C. SZEPESVÁRI : Bandit based Monte-Carlo planning. *Actes de European Conference on Machine Learning (ECML)*.

[13] T. LAI et H. ROBBINS : Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1):4–22, 1985.

[14] T. LATTIMORE et C. SZEPESVARI : *Bandit Algorithms*. Cambridge University Press, 2019.

[15] L. LI, W. CHU, J. LANGFORD et R. E. SCHAPIRE : A contextual-bandit approach to personalized news article recommendation. *Actes de International World Wide Web Conference (WWW)*, 2010.

[16] M. PUTERMAN : *Markov Decision Processes. Discrete Stochastic. Dynamic Programming.* Wiley, 1994.

[17] H. ROBBINS : Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952.

[18] C. RÉDA, E. KAUFMANN et A. DELAHAYE-DURIEZ : Machine learning applications in drug development. *Computational and Structural Biotechnology Journal*, 18:241–252, 2020.

[19] D. SILVER, A. HUANG, C. J. MADDISON, A. GUEZ, L. SIFRE, G. van den DRIESSCHE, J. SCHRITTWIESER, I. ANTONOGLOU, V. PANNEERSHELVAM, M. LANCTOT, S. DIELEMAN, D. GREWE, J. NHAM, N. KALCHBRENNER, I. SUTSKEVER, T. LILLICRAP, M. LEACH, K. KAVUKCUOGLU, T. GRAEPEL et D. HASSABIS : Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

[20] D. SILVER, J. SCHRITTWIESER, K. SIMONYAN, I. ANTONOGLOU, A. HUANG, A. GUEZ, T. HUBERT, L. BAKER, M. LAI, A. BOLTON, Y. CHEN, T. LILLICRAP, F. HUI, L. SIFRE, G. van den DRIESSCHE, T. GRAEPEL et D. HASSABIS : Mastering the game of Go without human knowledge. *Nature*, 550:354–, 2017.

[21] R. SUTTON et A. BARTO : *Reinforcement Learning : an Introduction*. MIT press, 2018.

[22] G. TESAURO : TD-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Comput.*, 6(2):215–219, 1994.

[23] W. THOMPSON : On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25:285–294, 1933.