Apprentissage par Renforcement

Lounès Meddahi
(lounes.meddahi@gmail.com)

Superviseur : Pr. Phillipe Preux

7 novembre 2021

Table des matières

Table des figures

1 Introduction

Le but de ce rapport est de présenter les principaux algorithmes d'apprentissage par renforcement. L'apprentissage par renforcement se distingue de tous les autres paradigmes (supervisé, non supervisé et semi-supervisé) car il consiste à étudier l'évolution d'un agent dans un environnement. Ces algorithmes seront étudiés par le biais de leurs implémentations [1] ainsi que leurs complexités algorithmique. Les algorithmes seront appliqués à un problème de décision de Markov afin de trouver le moyen d'optimiser une certaine fonction objectif grâce à des retours.

Les applications de l'apprentissage par renforcement sont multiples. Le projet BabyAI du Montreal Institute for Learning Algorithms (MILA), qui consiste en l'apprentissage de différentes tâches d'un agent récompensé à chaque succés. Le projet de chercheurs de l'université du Tennesse intitulé Reinforcement learning-based multi-agent system for network traffic signal control, qui a pour objectif de créer un contrôleur de feux de circulations afin de régler les problèmes d'embouteillages. Ou encore les projets de DeepMind Alpha-Go et Alpha-Go Zero qui sont des programmes jouant au jeu de GO.

D'abord, on étudiera quatre algorithmes sur la base du problème d'un chauffeur de taxi. Ensuite, la résolution d'un labyrinthe sera étudié pour les trois algorithmes suivants. Enfin, on conclura sur les limites de certains algorithmes et des perspectives possibles.

2 Processus de décision Markovien

Un processus de décision Markovien est un modèle aléatoire où un agent prend des décisions et où les résultats de ses actions sont aléatoires.

- un ensemble d'états , s ∈ S
- un ensemble d'actions possibles, a ∈ A
- P est la fonction de transition : *pour chaque couple (état, action), cette fonction indique la probabilité que le système soit ensuite dans chaque état* :

$$P : S \times A \times S \to [0, 1]$$

- R est la fonction de retour : *à valeur réelle, cette fonction formalise les conséquences d'une action émise dans un état.* :

$$R : S \times A \times S \to \mathbb{R}$$

- une politique déterministe : *à un état, la politique associe une action :*

$$\pi : S \to A$$

- une politique stochastique : *à un état, la politique associe une distribution de probabilités sur les actions :*

$$\pi : S \times A \to [0, 1]$$

- $V^{\pi}(s)$ est la valeur pour chaque état s d'un environnement pour une politique π : *c'est l'espérance de R quand l'environnement est initialement dans l'état s :*

$$V^{\pi}(s) = \mathbb{E}[Rs_0 = s]$$

- $Q^{\pi}(s, a)$ est la qualité d'une paire état action action (s,a) pour une politique π : *ça nous permet d'étudier l'action a une fois faite dans l'état s :*

$$Q^{\pi}(s, a) = \mathbb{E}[V^{\pi}(s_0) \mid s_0 = s, a_0 = a, a_{t>0} \sim \pi]$$

1. `https://github.com/LounesMD/Stage2021_RL`

3 Chauffeur de Taxi

Les quatre algorithmes suivants étudieront le problème d'un chauffeur de taxi qui doit choisir quelle action réaliser lorsqu'il arrive dans une nouvelle ville afin de maximiser ses gains. Pour cela, il y aura 3 villes possibles : A, B et C, et 3 actions possibles : $a1$, $a2$ et $a3$, qui lui donneront des gains différents.

3.1 Algorithme n°0 : calcul de la valeur

3.1.1 Idée générale

Les revenus possibles du chauffeur de taxi sont estimés grâce à une fonction valeur. La fonction valeur estime les revenus du chauffeur en fonction de la transition entre les différents états (i.e. les différentes villes). L'expression de cette fonction est donnée par l'équation de Bellman (1). On appellera γ le *facteur déprécié*.

$$V^{\pi}(s) = \sum_{a \in A} \pi(s,a) \sum_{s' \in S} P(s,a,s')[R(s,a,s') + \gamma V^{\pi}(s')] \tag{1}$$

Le calcul de la valeur pour chaque état et pour une politique donnée peut se faire de la manière suivante :

1. Initialiser : $V(s) \leftarrow 0$, $\forall s \in S$, $k \leftarrow 0$
2. Itérer : $V_{k+1}(s) = \sum_{a \in A} \pi(s,a) \sum_{s' \in S} P(s,a,s')[R(s,a,s') + \gamma V^{\pi}(s')]$; $k \leftarrow k+1$
3. Jusqu'à : $\| V_k - V_{k-1} \|_{\infty} \leq \epsilon$

3.1.2 Implémentation

Pour implémenter un tel algorithme, il est nécessaire d'utiliser une fonction de transition afin de savoir vers quel état l'agent va pouvoir se placer après une certaine action, une fonction de retour pour connaître les gains associés à chaque action exécutée et une politique afin de connaître quelles actions seront possibles. Ainsi, on va choisir notre valeur ϵ et effectuer l'algorithme d'écrit ci-dessus pour avoir en retour la valeur associée à chaque état en fonction de la politique donnée.

3.1.3 Résultats

L'application de cet algorithme avec une politique d'équiprobabilité du choix des actions indique que le revenu à long terme de l'agent sera en moyenne de $87,43$ s'il démarre dans la ville A, $98,6$ s'il démarre dans la ville B et $87,40$ s'il démarre dans la ville C.

Un représentation de la fonction valeur de toutes les politiques stationnaires et déterministes possibles du problème du chauffeur de taxi, pour $\gamma = 0,9$ est donnée par la figure 1.

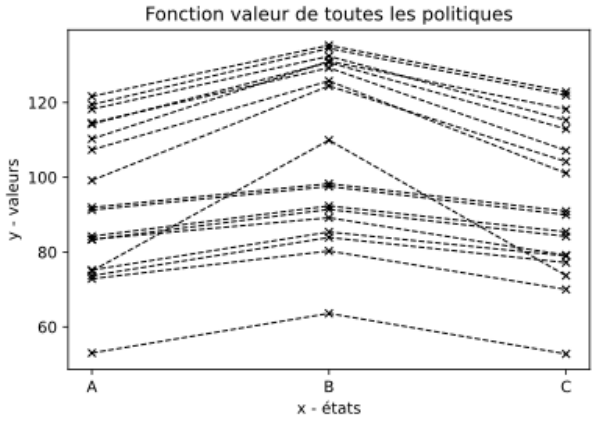


FIGURE 1 – Fonction valeur de toutes les politiques

La figure 1 indique que quelque soit la politique choisie par le chauffeur de taxi, il sera toujours préférable de commencer par la ville B. Cela s'explique par le fait que les transitions qui partent de B et ramène à B sont celles qui donnent le meilleur gain.

3.1.4 Complexité

Dans cet algorithme, on cherche à calculer la valeur associée à chaque état. Pour ce faire, on exécute des opérations jusqu'à converger asymptotiquement vers la valeur théorique de chaque état.

— Nombre d'opérations :

$$\sum_{\|V_{k+1}-V_k\|_\infty \geq \epsilon} \sum_{i=1}^{|S|} (\sum_{i=1}^{|A|} 1 \sum_{i=1}^{|S|} 3) = \sum_{\|V_{k+1}-V_k\|_\infty \geq \epsilon} 3|A||S|^2$$

Pour le cas $\epsilon = 10^{-6}$, il faut environ 130 opérations pour que $\| V_{k+1} - V_k \|_\infty \leq \epsilon$, soit un nombre total de $130(3 \times 3 \times 3 \times 3)$ opérations.

— L'espace nécessaire :
  — |S|×|A| pour la politique π
  — |A|×|S|^2 pour la fonction de transition $P(s, a, s')$
  — |A|×|S|^2 pour la fonction de retour $R(s, a, s')$
  — |S| pour stocker la valeur des états $V_k(s)$

Remarque : Dans le cas d'une implémentation avec des dictionnaires, il faudra ajouter le coût du stockage des clés.

3.2 Algorithme n°1 : itération sur les politiques

3.2.1 Idée générale

L'idée générale pour déterminer la valeur ϵ-optimale d'un environnement grâce à l'itération sur les politiques, est de générer une première politique aléatoire, puis de générer une politique meilleure que la précédente en essayant de trouver des meilleures valeurs, et ça, tant qu'on réussi à générer une politique strictement meilleure que la précédente. On trouvera ainsi pour tout ϵ une politique π telle que $\| V^\pi - V^* \|_\infty \leq \epsilon$. L'équation 2 sera utilisée pour calculer la valeur.

$$\sum_{s' \in S} P(s, a, s')[R(s, a, s') + \gamma V^\pi(s')] \tag{2}$$

Dans l'exemple du chauffeur de taxi, cela va lui permettre de connaître l'action à exécuter avant de changer de ville dans le but d'augmenter ses gains.

3.2.2 Implémentation

Comme précédemment, cet algorithme utilise une fonction de transition, de retour et une politique. Après avoir déterminé le seuil de précision ϵ et la première politique aléatoire, on cherchera la valeur ϵ-optimale associée à cette politique, puis pour chaque états vérifier s'il existe une action qui donne une plus grande valeur que celle retournée par la politique courante. Si oui, l'action associée à ces états sera modifiée afin d'améliorer la politique courante. Le processus est répété jusqu'à ce que deux politiques consécutives soient égales.

3.2.3 Résultats

Seulement 3 itérations de l'algorithme permet d'avoir la politique (déterministe) que le chauffeur de taxi doit suivre afin de maximiser ses gains.

Cette politique est donnée par : 'A' : 'a2', 'B' : 'a3', 'C' : 'a2'.

Les valeurs obtenus grâce à cette politique sont : 'A' : 121,65, 'B' : 135,30, 'C' : 122,83.

Le gain moyen sera de 121,65 s'il commence dans la ville A, 135,30 s'il commence dans la ville B et 122.83 s'il commence dans la ville C.

Cette politique déterministe peut-être vue comme une politique stochastique avec la répartition de probabilité suivante :

— politique['A'] = 'a1' : 0 , 'a2' : 1 , 'a3' : 0
— politique['B'] = 'a1' : 0 , 'a3' : 1
— politique['C'] = 'a1' : 0 , 'a2' : 1 , 'a3' : 0

3.2.4 Complexité

Dans cet algorithme, le but est de démarrer avec une politique quelconque puis de chercher une meilleur politique tant que c'est possible afin d'arriver sur la valeur optimale. Pour cela, deux boucles seront coûteuses en calculs. La première afin de déterminer la valeur ϵ-optimale, et la seconde pour déterminer si il existe pour chaque état une action meilleure que celle donnée par la politique courante.

— Nombre d'opérations :

$$\sum_{\pi_k \neq \pi_{k-1}} ((\sum_{\|V_i^{\pi_k} - V_{i-1}^{\pi_k}\|_\infty \geq \epsilon \frac{1-\gamma}{2\gamma}} \sum_{s \in S} \sum_{s' \in S} 3) + \sum_{s \in S} (arg\,\max_{a \in A} \sum_{s' \in S} 3))$$

$$= \sum_{\pi_k \neq \pi_{k-1}} ((\sum_{\|V_i^{\pi_k} - V_{i-1}^{\pi_k}\|_\infty \geq \epsilon \frac{1-\gamma}{2\gamma}} 3|S|^2) + 3|A||S|^2)$$

Avec $\epsilon = 0.01$, on trouve que :
— $(\sum_{\|V_i^{\pi_k} - V_{i-1}^{\pi_k}\|_\infty \geq \epsilon \frac{1-\gamma}{2\gamma}} 1) \approx 97$
— $(\sum_{\pi_k \neq \pi_{k-1}} 1) \approx 2$
Ce qui fait une complexité d'environ 5300 opérations.

— L'espace nécessaire :
— |S| pour la valeur V^{π_k}
— |A||S| pour la politique π
— $|A||S|^2$ pour la fonction de transition P(s, π_k ,s')
— $|A||S|^2$ pour la fonction de retour R(s,π_k,s')

3.3 Algorithme n°2 : itération sur la valeur

3.3.1 Idée générale

L'algorithme précédent utilise une première politique générée aléatoirement qui va être améliorée au fil des itérations afin d'arriver à une politique qui ne peut plus être améliorée. Ici, une équation calcule la valeur pour toutes les actions possibles pour chaque état et garde seulement la plus grande valeur. Cet algorithme d'itération sur les valeurs utilise l'équation d'optimalité de Bellman (3) pour calculer la valeur d'un état.

$$\max_{a \in A(s)} \sum_{s' \in S} P(s, a, s')[R(s, a, s') + \gamma V^{\pi}(s')] \tag{3}$$

3.3.2 Implémentation

L'implémentation de cet algorithme permet d'atteindre en deux itérations la politique optimale. La première itération donne la fonction valeur ϵ-optimale, et la seconde la politique associée à cette fonction valeur.

3.3.3 Résultats

On retrouve ainsi les mêmes résultats qu'avec l'algorithme précédent pour une politique (déterministe) :
'A' : 'a2', 'B' : 'a3', 'C' : 'a2'
Les valeurs associées à cette politique sont :
'A' : 121,65, 'B' : 135,30, 'C' : 122,83

3.3.4 Complexité

La complexité relative aux deux itérations est calculée comme suit. On note que l'espace mémoire est le même que l'algorithme précédent.

Nombre d'opérations :

$$\Big(\sum_{\|V_i^{\pi^k} - V_{i-1}^{\pi^k}\|_\infty \geq \epsilon \frac{1-\gamma}{2\gamma}} \Big(\sum_{s \in S}(|A| \sum_{s' \in S} 3)\Big) + \sum_{s \in S}(arg \max_{a \in A} \sum_{s' \in S} 3)$$

$$= \Big(\sum_{\|V_i^{\pi^k} - V_{i-1}^{\pi^k}\|_\infty \geq \epsilon \frac{1-\gamma}{2\gamma}} |S|(3|A||S|)\Big) + |S|(3|A||S|)$$

3.4 Algorithme n°3 : Différence temporelle (TD) pour une politique déterministe

3.4.1 Idée générale

L'algorithme TD permet d'obtenir une fonction valeur V qui converge vers la fonction valeur optimal V^* en s'appuyant sur la différence temporelle. La différence temporelle $\alpha(s, n(s))$ est un terme correctif à apporter à V pour converger asymptotiquement. Le but de cet algorithme est de s'approcher de la valeur optimale sans connaître les fonctions de transitions et de retours. La valeur associée à chaque état est calculé grâce à l'équation 4.

$$V^\pi(s_t) = V^\pi(s_t) + \alpha(s_t, n(s_t))[r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)] \text{ où } \alpha(s, n(s)) = \frac{1}{1 + n(s)} \tag{4}$$

3.4.2 Implémentation

La fonction valeur est initialisée à 0 pour tous les états et le taux d'apprentissage à 1. On déterminera l'état de départ, puis jusqu'à accéder à l'état final, une action est émise pour observer les conséquences (état d'arrivée et gain). Les valeurs associées aux états sont mises à jour et $n(s)$ incrémenté $0, 1$. A l'issu, le programme retournera la fonction valeur associée à la politique donnée.

3.4.3 Résultats

En appliquant cet algorithme à la politique optimale, les fonctions valeurs convergent bien les valeurs optimales. La figure 2 montre la convergence de la fonction valeur pour chaque état.

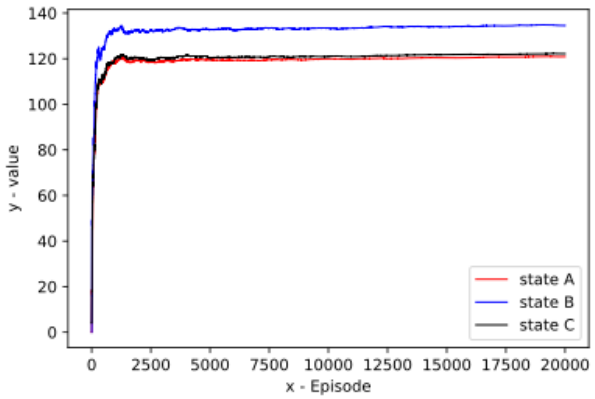


FIGURE 2 – Convergence de la valeur de chaque état

3.4.4 Complexité

A l'opposé des algorithmes précédents, l'algorithme TD converge vers les fonctions valeurs. Par conséquent, l'algorithme itérera autant fois qu'il le faut pour converger.

Nombre d'opérations :

La notation f_t définie que l'état final f a été atteint à l'instant t.

$$\sum_{\infty} \sum_{s_t \neq f_t} 6$$

L'espace nécessaire :
— |S| Pour la valeur V^{π}
— |S| Pour n(s)
— t pour stocker les retours r_t
— t+1 pour stocker les états s_{t+1}

On trouve une complexité de l'ordre de $\theta(2|S|+2t)$, pouvant être réduite si l'intégralité des retours et des états transitoires ne sont pas conservées.

4 Labyrinthe

Les algorithmes suivants sont utilisés pour trouver le plus court chemin vers la sortie d'un labyrinthe. Pour ce faire, un agent est placé dans cet environnement et doit l'explorer pour accéder à la sortie en un minimun d'actions. Il existe deux types d'algorithmes d'apprentissage par renforcement : *off-policy* et *on-policy*. Les algorithmes *off-policy* (ex. Q-Learning) ont pour particularité d'approcher l'équation de calcul en utilisant l'action estimée comme étant la meilleur possible. Les algorithmes *on-policy* (ex. SARSA) utilisent dans l'équation de calcul l'action qui est exécutée par l'agent. Les algorithmes off-policy peuvent ainsi utiliser des échantillons collectés par une politique autre que la politique courante pour mettre à jour la fonction de calcul.

4.1 Algorithme n°4 : Q-Learning

4.1.1 Idée générale

Q-Learning est un algorithme d'apprentissage par renforcement. Ce qui signifie que la politique, la fonction de transition et la fonction de retour ne sont plus connues car l'évolution de l'agent dans son environnement est observée.

Pour ce faire, l'algorithme effectuera des "épisodes" qui correspondent à la série d'actions réalisée par l'agent pour passer de l'état initial (ex. l'entrée d'un labyrinthe) à l'état final (ex. la sortie d'un labyrinthe). Le but étant de minimiser le nombre d'actions pour atteindre l'état final. Les actions effectuées au fil d'un épisode sont analysées pour évaluer leurs conséquences au sein de l'environnement. Au lieu de la fonction valeur qui indique s'il est pertinent de passer par cet état pour optimiser la fonction objectif, la fonction qualité est utilisée. Elle indique s'il est efficace d'exécuter une certaine action dans un certain état. Formellement, la fonction qualité utilise l'équation d'optimalité de Bellman (5) pour calculer la qualité Q^* d'un couple état-action.

$$Q^*(s_t, a_t) = \sum_{s' \in S} P(s_t, a_t, s')[R(s_t, a_t, s') + \gamma \max_{a' \in A} Q(s_{t+1}, a')] \tag{5}$$

Ce paradigme d'apprentissage implique certaines questions comme celui du choix de l'action à réaliser ou la mise oeuvre de la phase d'exploration.

4.1.2 Implémentation

Le choix des actions à réaliser est le point crucial de cet algorithme. Ce choix est effectué suivant différentes approches :

— **aléatoire** : Cette méthode consiste à choisir une action aléatoire lorsque l'agent arrive dans un nouvel état. Le problème de ce choix est qu'au premier épisode ou au dernier l'agent ne va jamais utiliser ce qu'il a déjà fait lors des épisodes précédents.

— **gloutonne** : L'agent va toujours choisir l'action qui permet d'avoir le meilleur retour à partir de son état actuel. L'action choisie sera ainsi $a_{gloutonne}(s_t) = arg\ max_{a \in A(s_t)}\ Q(s_t, a)$. Cependant, lorsqu'une première solution est trouvée la phase d'exploration est immédiatement arrêtée sans laisser la possibilité de trouver une solution plus optimale.

— **Boltzmann** : L'approche Boltzmann est un cas particulier de la séléction $softmax$. Cette sélection permet de donner une certaine distribution de probabilité aux différentes actions en fonction de leur qualité. La répartition de Boltzmann est donnée par l'équation 6.

$$Pr[a_t \mid s_t] = \frac{e^{\frac{Q(s_t,a_t)}{\tau}}}{\sum_{a \in A(s_t)} e^{\frac{Q(s_t,a_t)}{\tau}}} \tag{6}$$

On appelle τ un réel positif appelé température. Plus la température est grande, plus le choix des actions est proche d'une sélection aléatoire. A l'opposée, plus la température est faible plus le choix des actions sera similaire à un approche gloutonne. L'enjeu de cette méthode est de trouver la bonne manière de choisir ce τ.

4.1.3 Résultats

Pour cette étude, le labyrinthe décrit par la figure 3 a été utilisé. Les expérimentations réalisées se base sur 15 séries de 70 épisodes chacune et ça pour les différentes approches décrites précédemment. Les trois graphiques montrent illustrés par la figure 4 l'évolution du nombre de pas à faire atteindre la sortie du labyrinthe (i.e. la case bleue).

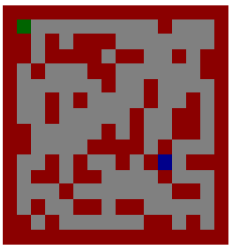
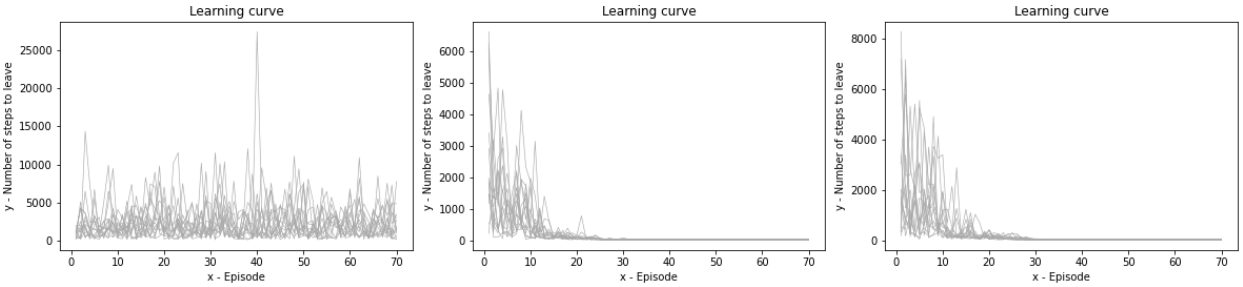


FIGURE 3 – Labyrinthe d'étude



(a) Fonction aléatoire (b) Fonction gloutonne (c) Fonction Boltzmann ($\tau = 1$)

FIGURE 4 – Efficacité des différentes approches

On constate donc que les méthodes gloutonne et Boltzmann permettent un apprentissage de l'environnement en faisant de moins en moins de pas pour arriver à l'état final, là où le choix aléatoire n'a aucune amélioration à chaque épisode. Le nombre de pas est initialement très grand car l'agent n'a aucune information sur le labyrinthe, puis le nombre de pas diminue jusqu'à trouver une trajectoire quasi-optimale pour terminer le labyrinthe.

4.1.4 Réflexions

Étudions maintenant l'application de l'algorithme Q-Learning dans un environnement non-stationnaire (i.e qui évolue au cours du temps).

I) Q-Learning avec une fonction qualité déjà entraînée

1) Déplacement de l'état final

Dans un environnement avec $\tau = 1$, la figure 5 montre que lorsque la sortie est déplacée l'agent va d'abord se diriger vers la sortie initiale. Ensuite, la fonction qualité est réajustée ce qui a pour conséquence la décroissance de la courbe d'apprentissage et la prise en compte en compte de la nouvelle sortie par l'agent.

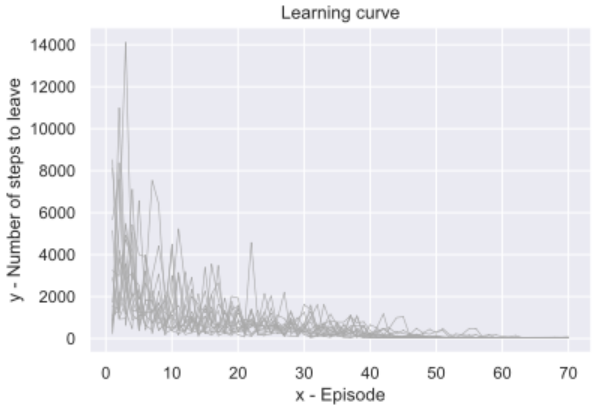
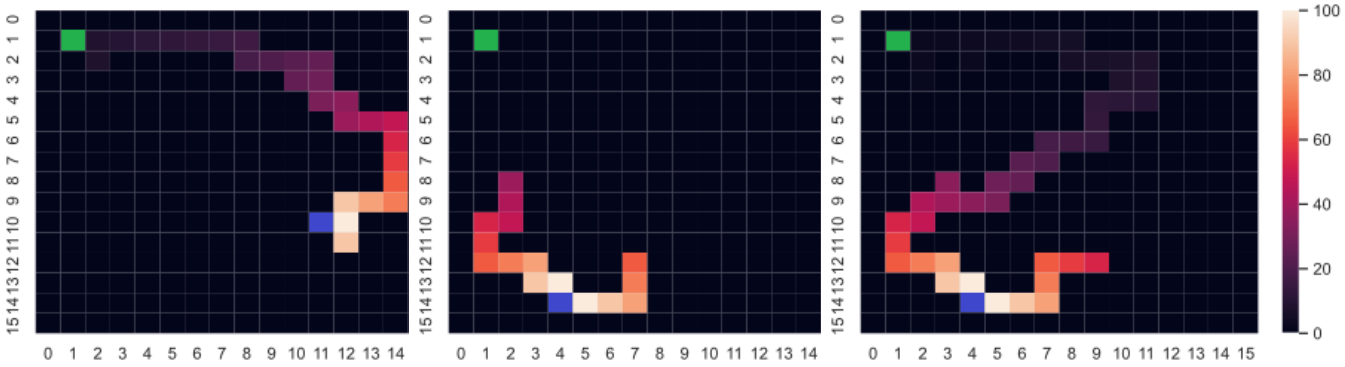


FIGURE 5 – Courbe d'apprentissage pour le réajustement de la fonction qualité

Une autre manière de voir ces résultats est d'utiliser une Heat Map (figure 6). On représente la fonction qualité sur chaque emplacement du labyrinthe en fonction de la valeur maximale de la fonction qualité sur cet emplacement. Un réajustement de la fonction qualité est ainsi observable.



(a) 70 épisodes passés dans le premier environnement

(b) 15 épisodes passés dans le nouvel environnement

(c) 70 épisodes passés dans le nouvel environnement

FIGURE 6 – Heat Map de la fonction qualité selon le nombre d'épisode passé dans un environnement

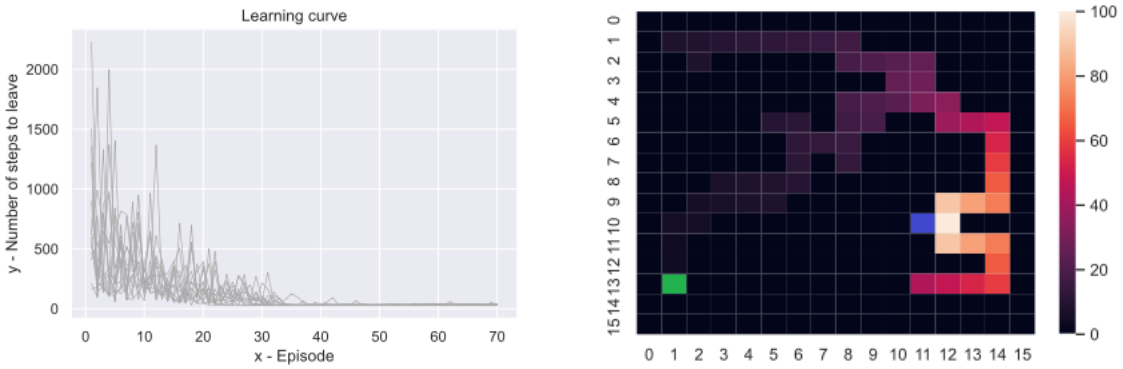
On peut donc conclure grâce à ces Heat Map que notre agent a *évolué* dans un nouvel environnement malgré qu'il connaissait une fonction qualité déjà entraînée. L'agent a donc continuer de mettre à jour cette fonction qualité afin de réajuster les valeurs pour former un chemin vers la nouvelle sortie.

2) Modification de l'état initial

Ici, la fonction qualité est aussi entraînée dans un premier environnement avant de déplacer la position de l'état initial. La figure 7 permet de constater que contrairement au cas précédent, la fonction qualité *s'ajuste* afin de former un nouveau chemin vers l'état final tout en conservant le chemin initial.

Bilan

L'étude de cette première partie permet de conclure que l'algorithme de Q-Learning permet un ***ajustement*** de la fonction qualité dans un environnement qui a évolué une fois. On peut désormais s'intéresser à un agent placé dans un environnement *non stationnaire*.



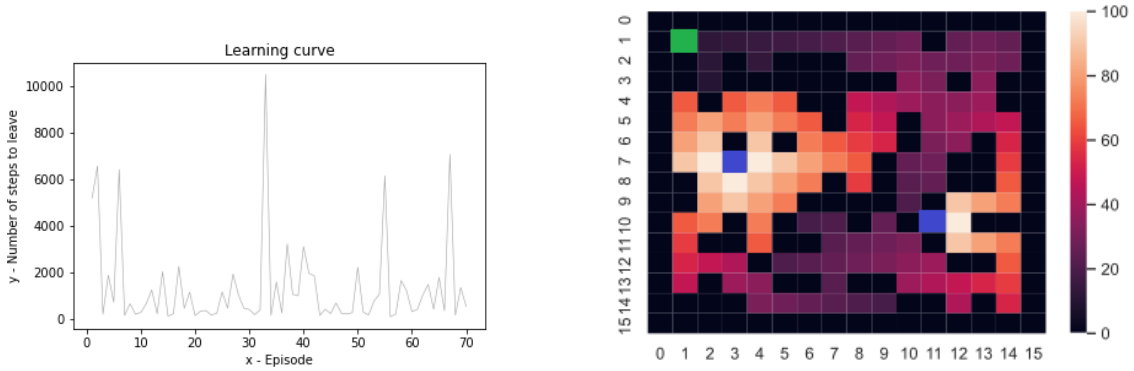
(a) Courbe d'apprentissage (b) Heat Map

FIGURE 7 – Informations pour un environnement après déplacement de l'état initial

II) Q-learning dans un environnement non stationnaire

1) Déplacement de la sortie à deux endroits différents

D'abord, on considère que l'état final alterne de position à chaque épisode. La figure 8a permet de constater que le nombre de pas à faire n'est jamais constant. L'explication est donnée par la figure 8b qui montre que la fonction qualité va plutôt agir comme s'il y avait toujours 2 états finaux. L'instabilité de la courbe d'apprentissage est donc justifiée par le fait que lorsque l'agent arrive à un endroit qui lui permet d'aller soit vers l'emplacement de l'état final A soit vers l'emplacement de l'état final B, l'agent a autant de chance d'aller vers l'une ou l'autre. Cette façon d'agir donne une fonction qualité inutilisable pour l'agent. La valeur de la température était de $\tau = 50$ afin d'éviter que l'agent soit bloqué lorsqu'il se trompe de chemin.



(a) Courbe d'apprentissage (b) Heat Map

FIGURE 8 – Évolution d'un agent placé dans un environnement non stationnaire

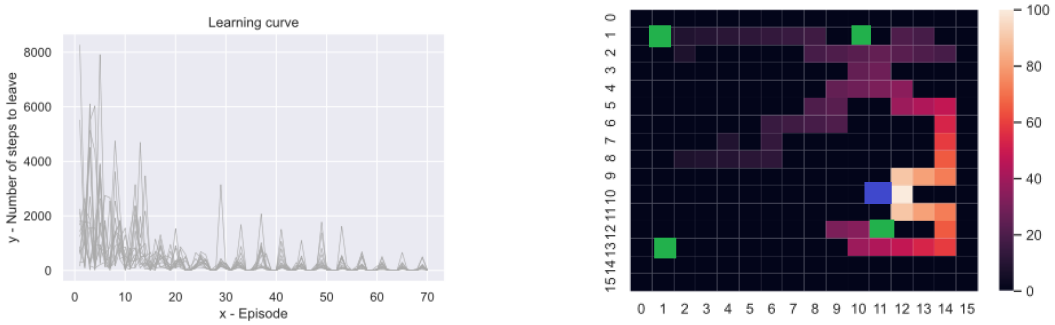
2) Modification du point de départ

La figure 9 illustre que l'algorithme permet d'ajuster la fonction qualité de telle sorte que des nouveaux chemins apparaissent pour relier tous les états initiaux vers l'état final (figure 9b). La conséquence est une monotonie de la learning curve (figure 9a). Les faibles variations ponctuelles sont justifiées par le fait que l'agent peut parfois se tromper de sens lorsque le chemin qu'il suit croise un nouveau chemin.

3) Déplacement aléatoire du point d'arriver

Enfin, le comportement de l'agent sera étudié lorsque l'état final se déplace de manière aléatoire dans l'environnement.

La figure 10 montre qu'il n'y a aucune régularité dans la courbe d'apprentissage. Cela s'explique par le fait que l'agent ne peut pas suivre le chemin formé par la fonction qualité au fil des épisodes



(a) Courbe d'apprentissage

(b) Heat Map de la fonction qualité après 70 épisodes

FIGURE 9 – Étude d'un agent dans un environnement qui change l'état initial à tous les tours (Les cases verte représentent les états initiaux possibles et la case bleu l'état final)

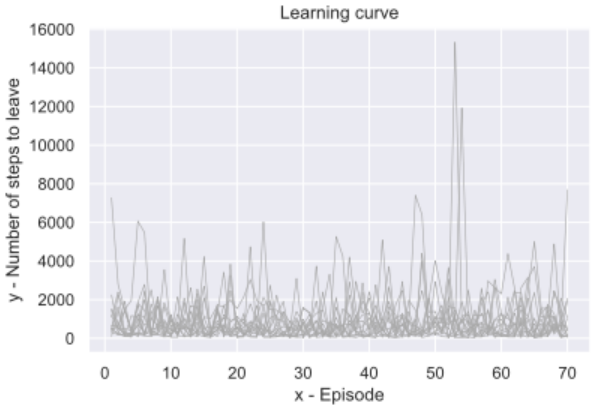


FIGURE 10 – Courbe d'apprentissage dans un environnement aléatoire

comme pour les algorithmes précédents.

4) Bilan et perspectives

Ces deux études permettent de discuter sur les limites de l'algorithme de Q-Learning avec un τ constant. En effet, un τ constant pour un environnement qui a une seule modification implique énormément d'actions à faire avant que l'agent réalise que l'environnement à évolué. Cependant, si l'environnement est modifié une première fois, un réajustement est effectuée sur la fonction de qualité de sorte que l'agent s'adapte à ce nouvel environnement. Aussi, la fonction qualité permet à Q-Learning de se réajuster positivement seulement si les modifications apportées à l'environnement ne vont pas modifier la fonction qualité à tous les épisodes au détriment des résultats des épisodes précédents. Et si l'environnement évolue aléatoirement en modifiant la fonction qualité, l'algorithme ne va pas savoir comment réagir.

Les conclusions précédentes nous permettent de proposer l'amélioration suivante :

— L'implémentation d'une fonction qui va apporter des modifications de τ à tous les tours. Faire décroître τ vers 0 si l'agent fait de moins en moins d'action pour arriver à l'état final à chaque épisode (cela va nous permettre de garder un bon nombre de pas pour un environnement stable). Ou bien, en faisant croître τ de plus en plus rapidement si on fait plus de pas que la moyenne des pas fait pour les épisodes précédents (cas de l'environnement non-stationnaire).

Une autre idée pourrait être d'utiliser un coefficient associé à chaque état de notre environnement qui permet de mesurer si notre environnement est stationnaire ou non. Cependant, il est actuellement impossible de savoir si c'est l'environnement qui évolue, ou bien dû au hasard que l'agent ne trouve pas l'état final.

Ces idées sont encore sujet de discussion et de recherche dans l'actualité de l'apprentissage par renforcement et ne seront donc pas traitées dans ce document.

4.1.5 Complexité

Un tel algorithme utilisant l'équation d'opitmalité de Bellman a pour complexité :
Nombre d'opérations :

$$\sum_{\infty} \sum_{s_t \neq f_t} 7 + |A|$$

4.2 Algorithme n°5 : State-Action-Reward-State-Action (SARSA)

4.2.1 Idée générale

SARSA est un algorithme *on-policy* par conséquent il estime la qualité de manière itérative en utilisant l'action exécutée au lieu de celle qui semble être la meilleur. L'équation 5 devient ainsi l'équation 7. Le reste de l'implémentation reste inchangée par rapport à l'algorithme de Q-Learning.

$$Q(s_t, a_t) = \sum_{s' \in S} P(s_t, a_t, s')[R(s_t, a_t, s') + \gamma Q(s_{t+1}, a_{t+1})] \tag{7}$$

4.2.2 Résultats

Rien n'est clair concernant les différences de performance entre les algorithmes *on-policy* et *off-policy*. La figure 11 montre que SARSA peut être aussi performant que l'algorithme de Q-Learning pour le problème du labyrinthe. Toute fois, comme ces algorithmes sont *on-policy* et qu'ils apprennent d'échantillons plus récents, on peut s'attendre a ce qu'ils soient plus performant pour des environnement non stationnaires.

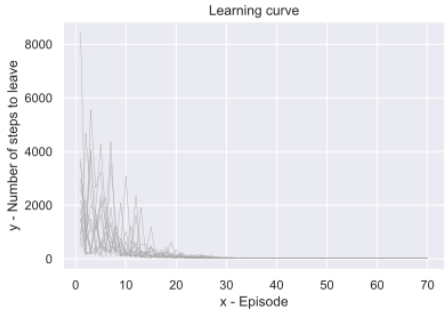


FIGURE 11 – SARSA : courbe d'apprentissage

4.2.3 Complexité

La mise oeuvre de SARSA est similaire à celle de Q-learning les complexités algorithmiques sont comparables. Il faut quand même prendre en compte que cet algorithme ne cherche pas l'action qui est potentiellement la meilleure.

Nombre d'opérations :

$$\sum_{\infty} \sum_{s_t \neq f_t} 7$$

4.3 Algorithme n°6 : Q(λ)

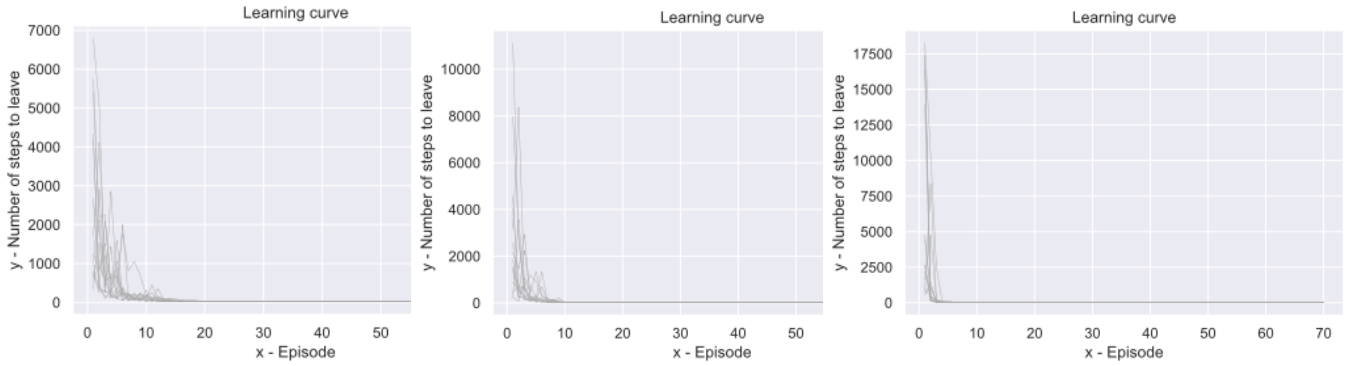
4.3.1 Idée générale

$Q(\lambda)$ s'appuie sur le principe de localité (i.e. le passé proche est une bonne approximation du futur proche). L'algorithme Q(λ) prend en compte ce principe pour mettre en place *l'éligibilité* e d'un couple état-action (s, a). Ainsi, l'action qui sera effectuée à un instant t mettra à jour $Q(s_t, a_t)$, mais aussi tous les couples état-action qui ont influencé le choix de l'action. L'équation qui permet de calculer la qualité est donnée par l'équation 8. La variable λ est appelée *coefficient d'éligibilité*.

$$Q(s_t, a_t) = Q(s_t, a_t) + \lambda[r_t Q(s_{t+1}, a^*) - Q(s_t, a_t)]e(s_t, a_t) \tag{8}$$

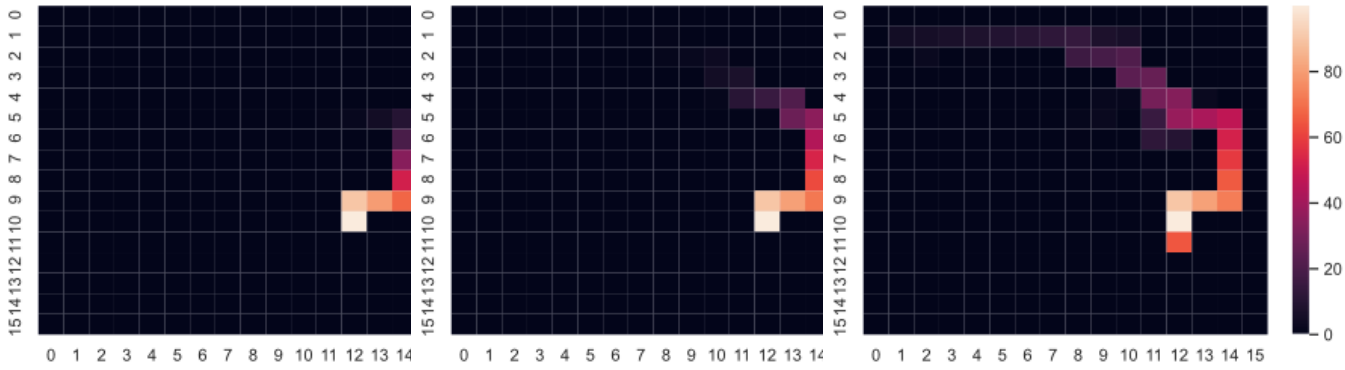
4.3.2 Résultats

La figure 12 montre l'application de l'algorithme $Q(\lambda)$ pour des valeurs croissantes de λ. On observe qu'à mesure que λ augmente que le seuil d'apprentissage est atteint plus rapidement. Cependant, le nombre de pas nécessaires aux première itérations augmentent lui aussi avec λ. La figure 13 représente différentes *Heat Map* pour des valeurs de λ différentes. On constate aussi qu'à mesure que λ augmente que le nombre d'épisodes nécessaires pour atteindre la sortie du labyrinthe diminue.



(a) $\lambda = 0.3$ (b) $\lambda = 0.6$ (c) $\lambda = 0.9$

FIGURE 12 – Q(λ) : courbe d'apprentissage



(a) $\lambda = 0.3$ (b) $\lambda = 0.6$ (c) $\lambda = 0.9$

FIGURE 13 – Q(λ) : Heat Map pour 5 épisodes passés dans l'environnement

4.3.3 Complexité

Ces algorithmes font beaucoup de calcul pour mettre à jour tous les états visités par l'agent. Nombre d'opérations :

$$\sum_{\infty}(\sum_{s_t \neq f_t} |A| + 5 + (\sum_{|S||A|} 5))$$

5 Conclusion

L'implémentation et l'étude des algorithmes classiques de l'apprentissage par renforcement présents dans la littérature scientifique a permis de poser les bases sur les notions de l'apprentissage par renforcement ainsi que sur certaines limites des ces algorithmes. L'étude des complexités a montré à quel point ces algorithmes sont coûteux en terme de capacité de calcul mais également en espace mémoire car il faut stocker toutes les valeurs calculées.

L'impact de l'apprentissage par renforcement dans les applications scientifiques est en lien direct avec les capacité de calcul disponible. Actuellement, ces capacités sont souvent très limitées mais nous pouvons penser que le développement future des ordinateurs quantiques dont la capacité de calcul est estimée être jusqu'à 100 millions de fois plus importante qu'un ordinateur classique, apporteront une solution durable à ce paradigme d'apprentissage.