Apprentissage par renforcement : revue théorique des algorithmes de Policy Evaluation et de Value Iteration

Alexandre TL

6 septembre 2022

Table des matières

Introduction **3**

I Chaînes de Markov **4**

1 Premières définitions et propriétés **4**
1.1 Cadre probabiliste 6

II Application : l'apprentissage par renforcement **7**

1 Introduction - Premières définitions **7**
1.1 Interaction agent-environnement 7
1.2 Etats, actions, et récompenses 8
1.3 Le but de l'apprentissage par renforcement 8

2 Modélisation mathématique du problème **9**
2.1 Chaîne de Markov comme modèle de l'environnement 9
2.2 Chaîne de décision markovienne (MDP) 9
2.3 Résoudre un MDP 9

3 Premier problème à résoudre : évaluer une politique **10**
3.1 Théorème du point fixe sur $\mathbb{R}^n$ 12
3.2 On revient au problème : convergence de l'algorithme d'évaluation d'une politique 17

4 Algorithme de Value Iteration **20**

5 Application : l'environnement Frozen Lake **25**
5.1 Implémentation Python 25
5.2 Environnement FrozenLake 26
5.3 FrozenLake, avec card(𝕊) variable 29

6 Conclusion et discussions **30**

Introduction

Ce document passe en revue la partie I du cours vidéo Intro RL disponible sur la chaîne Youtube Alexandre TL. Toutefois, les considérations faites ici permettent de justifier formellement les algorithmes mis en avant dans le cours : Policy Evaluation et Value Iteration.

Le premier paragraphe se verra être général : il s'agira notamment de donner la définition des chaînes de Markov, l'outil avec lequel les MDP pourront être définis. Ces MDP sont au coeur de l'apprentissage par renforcement car ils permettent de décrire formellement l'interaction entre l'agent et l'environnement.

Le second paragraphe réintroduit les idées et définitions majeures de l'apprentissage par renforcement : états, actions, récompenses, politique... Une section théorique démontrant un théroème du point fixe sur $\mathbb{R}^n$ permettra alors dans un second temps de démontrer la correction et la terminaison des algorithmes proposés.

Première partie
Chaînes de Markov

1 Premières définitions et propriétés

On commence par quelques définitions et propriétés pour poser le cadre probabiliste dans lequel nous nous plaçons, et voir comment les chaînes de Markov se manipulent.

Définition 1 *Une chaine de Markov à temps discret est une suite de variables aléatoires* $\{X_t\}_{t\geq 0}$ *à valeur dans un espace d'états telle que*

$$P(X_{t+1} = i_{t+1}|X_t = i_t,...,X_0 = i_0) = P(X_{t+1} = i_{t+1}|X_t = i_t)$$

On ne considèrera que des espaces d'états finis dans notre étude.

Définition 2 *On appelle matrice de transition en un pas la matrice* $P = (P_{ij})$ *la matrice dont les coefficients sont donnés par* $P_{ij} = P(X_{t+1} = j|X_t = i)$ *avec* i *et* j *des numéros associés à des états. On considérera dans la suite que les probabilités sont stationnaires, la chaine est alors dite homogène.*

Définition 3 *On appelle matrice de transition en* t *pas la matrice* $P^{(t)} = (P_{ij}^{(t)})$ *la matrice dont les coefficients sont donnés par* $P_{ij}^{(t)} = P(X_t = j|X_0 = i)$ *avec* i *et* j *des numéros associés à des états. C'est donc la probabilité d'obtenir l'état* j *au pas de temps* t*, sachant qu'on a commencé par l'état* i*.*

Proposition 1 *Pour tout* i,j*, on a*

$$0 \leq P_{ij}^{(t)} \leq 1$$

et également pour tout i *fixé on a*

$$\sum_j {P_{ij}}^{(t)} = 1$$

Démonstration
Soit $(i,j) \in [\![1,N]\!]^2$. Le scalaire $P_{ij}^{(t)}$ étant égal à une probabilité, on a directement l'inégalité. Soit $i \in [\![1,N]\!]$. Le scalaire $\sum_j {P_{ij}}^{(t)}$ représente la probabilité d'obtenir l'état j au pas de temps n en partant de i, avec j un état quelconque. Etant certain d'obtenir un état à ce pas de temps, il en vient que $\sum_j {P_{ij}}^{(t)} = 1$.

Proposition 2

$$\forall n \in \mathbb{N}, \quad P^n = P^{(n)}$$

Démonstration
On montre le résultat par récurrence sur $n \in \mathbb{N}$.

- L'égalité est vérifiée pour $n = 0$ (par définition de $P_{ij}^{(0)}$)et $n = 1$.
- On suppose la propriété vérifiée pour un certain $n \in \mathbb{N}$. On calcule :

$$\begin{aligned} P^{(n+1)} &= P^{(n)} P^{(1)} \quad \text{(cf Prop. 2 avec } k = n\text{)} \\ &= P^n P \\ &= P^{n+1} \end{aligned}$$

  ce qui montre la propriété au rang $n + 1$.
- Conclusion : la propriété est bien vérifiée pour tout $n \in \mathbb{N}$.

1.1 Cadre probabiliste

On se placera dans un espace probabilisé, à savoir un triplet (Ω, F, P) où Ω est l'univers, fini ou dénombrable.
F est une tribu de Ω, c'est-à-dire un ensemble de parties de Ω stable par passage au complémentaire, et représente l'ensemble des évènements.
P est une application de Ω dans [0,1] telle que $\sum_{\omega \in \Omega} \omega = 1$

L'espérance d'une variable aléatoire discrète est la somme des valeurs prises par la variable, pondérées par leur probabilité:

$$E(X) = \sum_{x_i \in X(\Omega)} x_i \ P(X = x_i)$$

On définit l'espérance conditionnelle d'une variable aléatoire, sachant un évènement B:

$$E(X|B) = \sum_{x_i \in X(\Omega)} x_i \ \frac{P(\{X = x_i\} \cap B)}{P(B)}$$

Seconde partie

Application : l'apprentissage par renforcement

1 Introduction - Premières définitions

Cette partie développe le domaine de l'apprentissage par renforcement. S'inspirant quelque peu de l'apprentissage animal, l'apprentissage par renforcement part de l'idée de l'apprentissage grâce à un signal extérieur. Chez les animaux, un tel apprentissage est en particulier possible grâce au noyau accumbens situé dans le cerveau. Un exemple de ce type d'apprentissage est par exemple celui d'un chien, qui aurait appris à associer l'action de s'assoir avec le fait de recevoir de la nourriture. L'apprentissage par renforcement crée des algorithmes qui se comportent de la même manière : ils sont capables d'associer certaines des **actions** qu'ils choisissent avec le fait de recevoir des **récompenses**. Ces récompenses sont des signaux que l'on envoie à l'algorithme afin de le guider dans la tâche qu'on veut qu'il réalise.

L'apprentissage par renforcement permet ainsi de créer des algorithmes capables de réaliser (ou plutôt d'apprendre à réaliser) une certaine tâche. L'exemple le plus populaire d'un tel algorithme est l'algorithme AlphaGo Zero, développé par DeepMind, qui a battu les plus grands joueurs mondiaux de go. L'algorithme, qu'on nomme aussi agent, exécute des actions (placer une pierre à un certain endroit) dans l'environnement que constitue le plateau de jeu. C'est seulement à la fin du match qu'il reçoit une récompense, positive si le match a été gagné, négative sinon. L'apprentissage d'un tel algorithme se fait donc par un entrainement, au cours duquel l'algorithme apprend à exécuter les *bonnes* actions pour maximiser les victoires.

1.1 Interaction agent-environnement

Il s'agit dans ce paragraphe de mettre en place les notations qui caractérisent l'interaction entre l'algorithme, ou bien l'agent, et l'environnement. Cette interaction se traduit par, à chaque pas de temps ($t = 0,1,2,...$), l'envoi par l'environnement à l'agent **d'un état** S_t et **d'une récompense** R_t, et l'envoi par l'agent à l'environnement **d'une action** A_t. C'est grâce à cet état et à cette récompense que l'agent va pouvoir décider de l'action à choisir dans le futur. La modélisation de cette interaction peut être représentée par la Figure 2.

C'est ensuite l'environnement, le chef d'orchestre, qui effectue la transition au temps $t+1$. Il envoie donc à l'agent S_{t+1} et R_{t+1}, et reçoit de la part de l'agent l'action choisie par l'agent A_{t+1}. Cette interaction se poursuit ainsi jusqu'à **l'horizon** $h \in \mathbb{N}$, le pas de temps maximal.

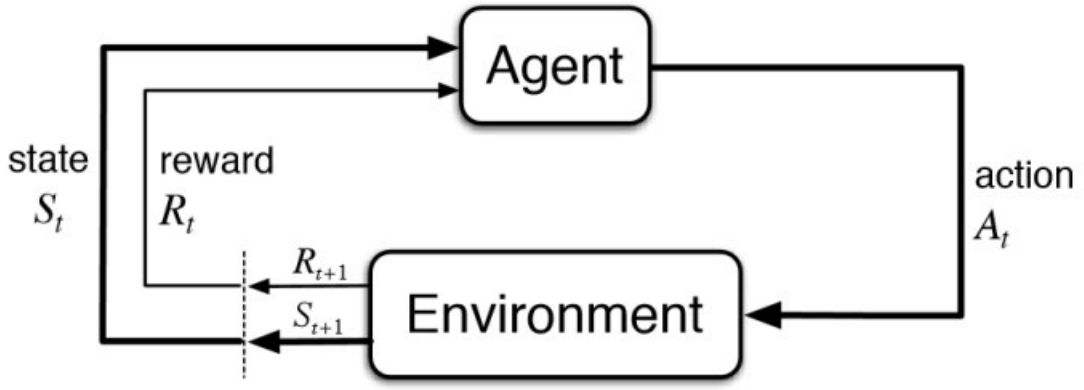

FIG. 1 – *L'interaction en apprentissage par renforcement. Crédits : R. Sutton*

1.2 Etats, actions, et récompenses

Les différents signaux abordés dans le paragraphe précédent permettent à l'agent de se repérer dans l'environnement:

- les états, notés avec la lettre s, permettent à l'agent de connaître exactement la situation de l'environnement avec lequel il interagit à un moment précis. De manière pratique, les états sont représentés sous forme de vecteur de nombres réels.
- les actions, notés avec la lettre a, sont les choix que peut faire l'agent afin de progresser dans l'environnement. Les actions sont représentées ici sous forme d'entier naturel (on parlera alors de l'action 0, l'action 1 etc...)
- les récompenses, notés avec la lettre r, sont des scalaires donnés par l'environnement qui permettent à l'agent d'obtenir une sorte d'évaluation sur l'état dans lequel il se trouve.

Les variables aléatoires S_t, A_t, R_t renvoient respectivement à l'état, à l'action et à la récompense obtenue par l'agent au pas de temps t.

1.3 Le but de l'apprentissage par renforcement

L'apprentissage par renforcement consiste alors à développer des algorithmes capables de décider la meilleure série d'actions à exécuter afin de maximizer les récompenses obtenues lors de l'interaction avec un certain environnement donné.

2 Modélisation mathématique du problème

2.1 Chaîne de Markov comme modèle de l'environnement

On définit dans un premier temps la chaîne de Markov associée à un certain environnement sur lequel on veut travailler. On considère donc $\mathbb{S}$ l'ensemble fini des états dans lequel l'agent peut se trouver dans l'environnement. Cet ensemble est donc aussi l'ensemble des états de la chaîne de Markov associée à l'environnement considéré. On bénéficie donc de la **propriété markovienne** :

$$\forall t \in \mathbb{N}, \forall (s_0,...,s_{t+1}) \in \mathbb{S}^{t+1}, P(S_{t+1} = s_{t+1} | S_t = s_t,...,S_0 = s_0) = P(S_{t+1} = s_{t+1} | S_t = s_t)$$

Ainsi, la connaissance de l'état présent est suffisante pour prédire entièrement l'avenir. Les états futurs ne dépendent donc que de l'état présent actuel.

2.2 Chaîne de décision markovienne (MDP)

On rajoute aux états de la chaîne de markov des actions et des récompenses. On note $\mathbb{A}$ l'ensemble des actions exécutables par l'agent. Il en vient que les états futurs dépendent de l'état actuel S_t et aussi de l'action choisie A_t (qui est, comme S_t, une variable aléatoire). Pour les récompenses, elles sont déterminés par S_t et A_t. En fait, on définit la fonction de récompense R:

$$R : \begin{cases} \mathbb{S} \times \mathbb{A} & \to & \mathbb{R} \\ (s,a) & \mapsto & R(s,a) \end{cases}$$

avec $R(s,a)$ une donnée propre à l'environnement. Ainsi, si l'agent se trouve dans l'état s_0 et qu'il exécute l'action a_0, il recevra à l'instant suivant la récompense $R(s_0,a_0)$ (fixée, connue à l'avance).

Alors, les ensembles $\mathbb{S}$, $\mathbb{A}$, les probabilités de transition entre chaque état et l'application R définissent une chaîne de décision markovienne, aussi appelée MDP (pour Markov Decision Process en anglais). Elle permet de modéliser un environnement donné.

2.3 Résoudre un MDP

Ainsi, on va pouvoir décrire formellement le but visé par un agent avec un environnement donné : celui de résoudre le MDP associé à l'environnement.

On définit la variable aléatoire du **retour** G_t au temps t par :

$$G_t = \sum_{k=0}^{h} \gamma^k R_{t+k+1}$$

avec $\gamma \in [0,1[$ le facteur de remise, h l'horizon. γ décrit à quel point on souhaite privilégier les récompenses proches dans le temps par rapport à celles qui se trouvent

plus loin dans le futur. L'horizon h est le pas de temps maximal. Ces deux quantités sont associés à un environnement donné.

Pour décrire un agent, on définit sa politique, qui associe à chaque couple (s,a) la probabilité de choisir l'action a, sachant que l'on se trouve dans l'état s. On a donc :

$$\pi : \begin{cases} \mathbb{S} \times \mathbb{A} & \to & [0,1] \\ (s,a) & \mapsto & \pi(a|s) \end{cases}$$

Pour que π soit bien définie en tant que politique, il faut donc que

$$\forall s \in \mathbb{S},\ \sum_{a \in \mathbb{A}} \pi(a|s) = 1$$

On dira alors que l'agent **suit** la politique π.

On peut enfin formaliser notre objectif : résoudre un MDP, c'est trouver une politique π_* qui maximise le retour G_t pour tout t. Dans le cadre de ce document, on considère connues la fonction récompense R, ainsi que les probabilités de transitions:

$$\forall t \in \mathbb{N}, \forall (s,a,s') \in \mathbb{S} \times \mathbb{A} \times \mathbb{S},\ P(S_{t+1} = s' \mid S_t = s,\ A_t = a) \textbf{ est connue}$$

On pourra, pour alléger les notations, noter $P(s'|s,a)$ pour désigner la quantité $P(S_{t+1} = s'|S_t = s, A_t = a)$. Une discussion sera ouverte à la fin de cette partie quant à la résolution d'un MDP sans cette connaissance.

3 Premier problème à résoudre : évaluer une politique

Dans le but de trouver une politique π_* qui maximise le retour G_t pour tout t, nous allons travailler itérativement. On va ainsi partir d'une première politique (en pratique, naïve) que l'on va **améliorer** à chaque itération. Pour s'assurer qu'une amélioration s'effectue, il faut pouvoir évaluer une certaine politique. Alors, on pourra évaluer la première politique et celle issue de la première itération et s'assurer que la dernière est bien meilleure que la première. On définit pour cela la **fonction de valeur** v_π associée à la politique π par :

$$\forall t \in \mathbb{N},\ \forall s \in \mathbb{S},\ v_\pi(s) = \mathbb{E}_\pi\big[G_t | S_t = s\big]$$

avec $\mathbb{E}_\pi$ qui désigne une espérance, indicée par π pour souligner le fait que c'est l'espérance du retour obtenu **en suivant la politique** π. On voit ainsi v_π comme un élément de $\mathbb{R}^{\text{card}(\mathbb{S})}$. On peut développer cette définition grâce à la connaissance des probabilités de transitions et des récompenses de l'environnement :

$$\begin{aligned}\forall t \in \mathbb{N}, \forall s \in \mathbb{S}, v_\pi(s) &= \mathbb{E}_\pi\big[R_{t+1} + \gamma G_{t+1} | S_t = s\big] \quad \text{(décomposition de } G_t \text{ avec Chasles)} \\ &= \mathbb{E}_\pi\big[R_{t+1} | S_t = s\big] + \gamma \mathbb{E}_\pi\big[G_{t+1} | S_t = s\big] \quad \text{(linéarité de l'espérance)} \\ &= \sum_{a \in \mathbb{A}} \pi(a|s) R(s,a) + \gamma \sum_{a \in \mathbb{A}} \pi(a|s) \sum_{s' \in \mathbb{S}} P(s'|s,a) v_\pi(s')\end{aligned}$$

En effet, pour la première somme, la récompense R_{t+1} ne dépend que l'état et de l'action prise au temps t. On calcule donc son espérance en fonction de l'action choisie. Quant à la seconde somme, il s'agit de regarder le futur en fonction de l'action prise à l'instant t. Pour chaque action, on aura une certaine probabilité, $P(s'|s,a)$, d'atterrir dans l'état s' (avec s' qui décrit $\mathbb{S}$). De là, on connait la fonction de valeur en cet état : $v_\pi(s')$. Il est intéressant de visualiser la situation avec l'arbre suivant:

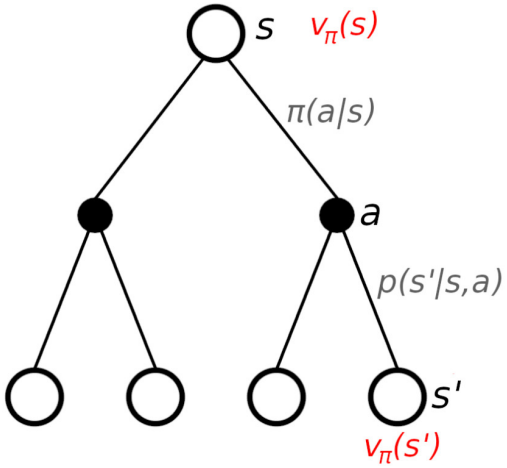

FIG. 2 – *A partir de l'état s, on regarde les possibilités futures pour l'état suivant s'.*

Cette équation sur v_π établie est appelée **équation de Bellman**. Il en vient alors un système linéaire de card($\mathbb{S}$) équations à card($\mathbb{S}$) inconnues. On peut le résoudre analytiquement pour calculer v_π. Cependant, cette résolution n'est pas généralisable pour la suite. Ainsi, on va choisir de le résoudre de façon itérative avec l'algorithme suivant.

On va alors construire un algorithme, l'algorithme d'ÉVALUATION DE POLITIQUE, pour calculer v_π avec π une certaine politique donnée. Cet algorithme va partir d'une estimation de v_π (quelconque) et va itérativement lui appliquer un certain opérateur L_π défini par:

$$L_\pi : \begin{cases} \mathbb{R}^{\text{card}(\mathbb{S})} & \to & \mathbb{R}^{\text{card}(\mathbb{S})} \\ V & \mapsto & R_\pi + \gamma T_\pi V \end{cases}$$

avec R_π le vecteur colonne de longueur card($\mathbb{S}$) qui contient, pour chaque état s, la récompense que l'on obtient en moyenne avec π en sortant de cet état:

$$\sum_{a \in \mathbb{A}} \pi(a|s) R(s,a)$$

et T_π la matrice de transition à un pas de la politique π. En notant $\mathbb{S} = \{s_1,...,s_N\}$, on a pour T_π :

$$\forall (i,j) \in [\![1,N]\!]^2, (T_\pi)_{i,j} = P(S_{t+1} = s_j |\ S_t = s_i)$$

Etant donné π, on peut calculer R_π et T_π grâce à notre connaissance des dynamiques de l'environnement.

Il est intéressant de remarquer que $L_\pi(V)$ est une réécriture matricielle du membre de droite de l'égalité de $v_\pi(s)$.

Le but du prochain paragraphe est alors d'utiliser un théorème du point fixe pour montrer que l'itération de L_π garantit une convergence de l'algorithme d'ÉVALUATION DE POLITIQUE.

Une fonction de valeur sera donc représentée par un élément de $\mathbb{R}^{\text{card}(\mathbb{S})}$. Pour $s \in \mathbb{S}$, on notera $V(s)$ la valeur de l'état s pour V (on prend une notation de fonction car V se rapproche de l'idée d'une fonction).

3.1 Théorème du point fixe sur $\mathbb{R}^n$

On fixe dans ce paragraphe un $n \in \mathbb{N}^*$

Définition 1 *Norme infinie sur $\mathbb{R}^n$*
*Pour $x = (x_1,...,x_n) \in \mathbb{R}^n$, on définit la **norme infinie** de x, notée $||x||_\infty$ par :*

$$||x||_\infty = \max_{i \in [\![1,n]\!]} |x_i|$$

Définition 2 *Suite et convergence dans $\mathbb{R}^n$*
On considère $(u_n) \in (\mathbb{R}^n)^\mathbb{N}$, une suite d'éléments de $\mathbb{R}^n$, et $l \in \mathbb{R}^n$. On dit que u converge vers l si :

$$\forall \varepsilon > 0, \exists N \in \mathbb{N}, \forall n \geq N, ||u_n - l||_\infty < \varepsilon$$

Définition 3 *Suite de Cauchy dans $\mathbb{R}^n$*
*On considère $(u_n) \in (\mathbb{R}^n)^\mathbb{N}$. On dit que u est **de Cauchy** pour la norme infinie si*

$$\forall \varepsilon > 0, \exists N \in \mathbb{N}, \forall p \geq N, \forall q \geq N, ||u_p - u_q||_\infty < \varepsilon$$

autrement dit, u est de Cauchy si tous les termes de la suite sont aussi proches que l'on souhaite pour la distance infinie (distance associée à la norme infinie), à partir d'un certain rang.

Proposition 1 *Toute suite de Cauchy de* $\mathbb{R}$ *converge dans* $\mathbb{R}$.

Démonstration
Soit $(u_n) \in \mathbb{R}^{\mathbb{N}}$ une suite de Cauchy.
Montrons dans un premier temps que (u_n) est bornée. On utilise la définition d'être de Cauchy pour (u_n) avec $\varepsilon = 1$. Il existe donc $N_1 \in \mathbb{N}$ tel que:

$$\forall p \geq N_1, \forall q \geq N_2, |u_p - u_q| \leq 1 \quad (\text{sur } \mathbb{R}, ||.||_\infty = |.|)$$

On prend $n \geq N_1$. Il vient alors que

$$|u_n - u_{N_1}| \leq 1$$

Par inégalité triangulaire avec la valeur absolue $|.|$, il vient :

$$|u_n| + |u_{N_1}| \leq |u_n - u_{N_1}| \leq 1$$

ou encore

$$|u_n| \leq 1 - |u_{N_1}|$$

On pose alors $d = 1 - |u_{N_1}|$, et

$$M = \max(u_1, ..., u_{N_1 - 1}, d)$$

il vient alors que

$$\forall n \in \mathbb{N}, |u_n| \leq M$$

ce qui montre que (u_n) est bornée.

Montrons alors que (u_n) converge. D'après le théorème de Bolzano-Weierstrass, on peut extraire de u une suite qui converge. Il existe donc $\varphi : \mathbb{N} \to \mathbb{N}$ strictement croissante telle que $(u_{\varphi(n)})$ converge vers $l \in \mathbb{R}$. Montrons que u converge vers l.

Soit $\varepsilon > 0$. Par définition de la convergence de $(u_{\varphi(n)})$ avec $\frac{\varepsilon}{2}$, il existe $N_0 \in \mathbb{N}$ tel que

$$\forall n \geq N_0, |u_n - l| \leq \varepsilon$$

Par définition d'être de Cauchy pour (u_n) avec $\frac{\varepsilon}{2}$, il existe $N_1 \in \mathbb{N}$ tel que

$$\forall p \geq N_1, \forall q \geq N_1, |u_p - u_q| \leq \varepsilon$$

Soit $n \geq \max(N_0, N_1)$. On a

$$\begin{aligned} |u_n - l| &= |u_n - u_{\varphi(n)} + u_{\varphi(n)} - l| \\ &\leq |u_n - u_{\varphi(n)}| + |u_{\varphi(n)} - l| \quad \text{(inégalité triangulaire)} \\ &\leq \frac{\varepsilon}{2} + \frac{\varepsilon}{2} \quad (\text{car } \varphi(n) \geq n) \\ &= \varepsilon \end{aligned}$$

ce qui montre que u converge vers l.

Proposition 2 *Toute suite de Cauchy de $\mathbb{R}^n$ converge dans $\mathbb{R}^n$*

Démonstration
On montre le résultat par récurrence sur $n \in \mathbb{N}^*$.

- $\mathbb{R}^1$ vérifie la propriété **au rang 1** (cf Prop. 4)
- **On suppose la propriété vérifiée** pour un certain $n \in \mathbb{N}^*$.
  Montrons que $\mathbb{R}^{n+1} = \mathbb{R}^n \times \mathbb{R}$ vérifie la propriété.
  On note $|.|$, $||.||_{n,\infty}$, $||.||_{n+1,\infty}$ les normes infinies sur les ensembles respectifs $\mathbb{R}$, $\mathbb{R}^n$, $\mathbb{R}^{n+1}$. (sur $\mathbb{R}$, il s'agit de la valeur absolue $|.|$).
  On peut réécrire cette dernière en fonction des deux premières, pour $x = (\mu, x_1)$ un élément de $\mathbb{R}^{n+1}$ (avec x_1 un réel):

$$||x||_{n+1,\infty} = \max\Big(||\lambda||_{n,\infty}, |x_1|\Big) \qquad (*)$$

  On considère $(u_n) \in (\mathbb{R}^{n+1})^{\mathbb{N}}$ de Cauchy.
  Ainsi, il existe $(v_n) \in (\mathbb{R}^n)^{\mathbb{N}}$ et $(w_n) \in \mathbb{R}^{\mathbb{N}}$ tel que

$$(u_n) = ((v_n,w_n))$$

  Soit $(p,q) \in \mathbb{N}^2$. On a, avec $(*)$:

$$||v_p - v_q||_{n,\infty} \leq ||u_p - u_q||_{n+1,\infty}$$

  ce qui montre que la distance entre chaque terme de la suite (v_n) est plus petite que la distance entre chaque terme de la suite (u_n). (u_n) étant de Cauchy, on en déduit que (v_n) est elle aussi de Cauchy, mais dans l'ensemble $\mathbb{R}^n$, pour $||.||_{n,\infty}$. On montre de la même manière que (w_n) est de Cauchy dans l'ensemble $\mathbb{R}$ pour $|.|$.

  Ainsi, par hypothèse de récurrence sur $\mathbb{R}^n$ et $\mathbb{R}$, il existe $l_v \in \mathbb{R}^n$ tel que (v_n) converge vers l_v, et il existe $l_w \in \mathbb{R}$ tel que (w_n) converge vers l_w.
  On pose $l_u = (l_v, l_w)$. Montrons alors que (u_n) converge vers (l_u).
  Soit $\varepsilon > 0$. Par convergence de u et v, il existe (N_0, N_1) des naturels tels que

$$\forall n \geq N_0, ||v_n - l_v||_{n,\infty} < \varepsilon$$

et

$$\forall n \geq N_1, |w_n - l_w| < \varepsilon$$

ainsi, en posant $N = \max(N_0, N_1)$ et en prenant $n \geq N$, il vient

$$||v_n - l_v||_{n,\infty} < \varepsilon \quad \text{et} \quad \forall n \geq N_1, |w_n - l_w| < \varepsilon$$

d'où

$$\max\Big(||v_n - l_v||_{n,\infty}, |w_n - l_w|\Big) = ||u_n - l_u||_{n+1,\infty} < \varepsilon$$

ce qui montre ainsi la convergence de (u_n) dans $\mathbb{R}^{n+1}$. La propriété est donc vérifiée au rang $n+1$.

– **Conclusion :** pour tout $n \in \mathbb{N}^*$, toute suite de Cauchy de $\mathbb{R}^n$ converge dans $\mathbb{R}^n$.

Définition 4 *On considère $f : \mathbb{R}^n \to \mathbb{R}^n$, $a \in \mathbb{R}^n$. On dit que f est **continue en** a si*

$$\forall \varepsilon > 0, \exists \delta > 0, \forall x \in \mathbb{R}^n \ \big|\big|x - a\big|\big|_\infty < \delta \implies \big|\big|f(x) - f(a)\big|\big|_\infty < \varepsilon$$

*On dit que f est **continue** si elle est continue pour tout $a \in \mathbb{R}$.*

Proposition 3 *Soit $f : \mathbb{R}^n \to \mathbb{R}^n$ contractante pour la norme infinie. Alors f est continue.*

Démonstration
Pour $\varepsilon > 0$ donné, on peut prendre $\delta = \frac{\varepsilon}{k}$, avec f k-lipschitzienne.

Proposition 4 *Soit $f : \mathbb{R}^n \to \mathbb{R}^n$ continue, u une suite d'éléments de $\mathbb{R}^n$ qui converge vers $a \in \mathbb{R}^n$, alors la suite $(f(u_n))$ converge vers $f(a)$.*

Démonstration
On utilise la définition de convergence de f puis celle de u.

Proposition 5 *Un théorème du point fixe dans $\mathbb{R}^n$. Soit f une fonction contractante de $\mathbb{R}^n$ dans $\mathbb{R}^n$. Alors f admet un unique point fixe α. De plus, si on définit la suite u par*

$$\begin{cases} u_0 = x_0 \in \mathbb{R}^n \\ u_{n+1} = f(u_n), \forall n \in \mathbb{N} \end{cases} \tag{1}$$

alors u converge vers α.

Démonstration

f étant contractante pour la norme infinie, on dispose de $k \in [0,1[$ tel que f soit k-lipschitzienne.

Montrons l'**unicité**. On suppose que α et β soient deux points fixes de f. Etant k-lipschitzienne, on a alors

$$||\alpha - \beta||_\infty = ||f(\alpha) - f(\beta)||_\infty \leq k||\alpha - \beta||_\infty$$

d'où $||\alpha - \beta||_\infty = 0$, et donc $\alpha = \beta$.

Montrons l'**existence**. Comme $f(\mathbb{R}^n) \subset \mathbb{R}^n$, la suite u est bien définie. Pour $n \in \mathbb{N}^*$, on a

$$||u_{n+1} - u_n||_\infty = ||f(u_n) - f(u_{n-1})||_\infty \leq k||u_n - u_{n-1}||_\infty$$

et donc, par récurrence sur $n \in \mathbb{N}$, il vient que

$$||u_{n+1} - u_n||_\infty \leq k^n||u_1 - u_0||_\infty \quad (*)$$

Montrons alors que u est de Cauchy. Soit $(n,l) \in (\mathbb{N}^*)^2$. On a :

$$\begin{aligned} ||u_{n+l} - u_n||_\infty &= \sum_{i=0}^{l-1} ||u_{n+i+1} - u_{n+i}||_\infty \quad \text{(inégalité triangulaire appliquée } l-1 \text{ fois)} \\ &\leq ||u_1 - u_0||_\infty \sum_{i=0}^{l-1} k^{n+i} \quad \text{(avec } (*)\text{)} \\ &\leq ||u_1 - u_0||_\infty \frac{k^n}{1-k} \quad (k \in [0,1[) \end{aligned}$$

ainsi, pour n assez grand, la distance entre u_{n+l} et u_n tend vers 0 : on en conclut que u est de Cauchy. Or, étant une suite d'éléments de $\mathbb{R}^n$, on sait, d'après la Prop. 9.1.2, que u converge dans $\mathbb{R}^n$. On note α cette limite. Pour tout $n \in \mathbb{N}$, on a

$$u_{n+1} = f(u_n)$$

comme f est contractante, elle est en particulier continue. (cf Prop. 9.1.3). Ainsi, en passant à la limite dans l'égalité précédente, il vient:

$$\alpha = f(\alpha)$$

ce qui montre donc l'existence d'un point fixe pour f.

3.2 On revient au problème : convergence de l'algorithme d'évaluation d'une politique

On définit l'algorithme d'ÉVALUATION DE POLITIQUE de la manière suivante:

Algorithm 1: Evaluation de politique

Entrée: π
Créer $V(s) = 0$ pour chaque $s \in \mathbb{S}$

repeat
  foreach s **do**
    $V(s) \leftarrow \sum_a \pi(a|s)R(s,a) + \gamma \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)V(s')$
  end
until *V ne change presque plus*

return $V \approx v_\pi$

Proposition 6 *L_π est contractante pour la norme infinie de $\mathbb{R}^{card(\mathbb{S})}$, i.e. il existe $k \in [0,1[$ tel que*

$$\forall (V_1, V_2) \in (\mathbb{R}^{card(\mathbb{S})})^2, \left|\left|L_\pi(V_1) - L_\pi(V_2)\right|\right|_\infty \leq k \left|\left|V_1 - V_2\right|\right|_\infty$$

Démonstration
Soit $((V_1, V_2) \in (\mathbb{R}^{\text{card}(\mathbb{S})})^2)$. On a

$$\begin{aligned}
\left|\left|L_\pi(V_1) - L_\pi(V_2)\right|\right|_\infty &= \left|\left|R_\pi + \gamma T_\pi V_1 - (R_\pi + \gamma T_\pi V_2)\right|\right|_\infty \\
&= \left|\left|\gamma T_\pi (V_1 - V_2)\right|\right|_\infty \\
&\leq \gamma \left|\left|T_\pi\right|\right|'_\infty \left|\left|V_1 - V_2\right|\right|_\infty \quad (\gamma \in [0,1[\ ) \\
&\leq \gamma \left|\left|V_1 - V_2\right|\right|_\infty
\end{aligned}$$

car T_π est stochastique, donc sa norme infinie (norme matricielle infinie de $\mathcal{M}_{\text{card}(\mathbb{S})}(\mathbb{R})$) est majorée par 1.

Proposition 7 *L'algorithme d'*ÉVALUATION DE POLITIQUE *converge, vers* v_π.

Démonstration
L_π est contractante pour la norme infinie sur l'espace vectoriel $\mathbb{R}^{\text{card}(\mathbb{S})}$, qui, d'après la proposition 5, est un espace vectoriel complet. Ainsi, on peut utiliser le théorème du point fixe (proposition 6) avec L_π. Cette application admet donc un unique point fixe $V_* \in \mathbb{R}^{\text{card}(\mathbb{S})}$. L'algorithme se termine donc.

De plus, on remarque que v_π vérifie les équations de Bellman, ce qui se traduit par le fait que

$$L_\pi(v_\pi) = v_\pi$$

ou encore, que v_π est un point fixe de L_π. Par unicité de ce point fixe, il en vient que

$$V_* = v_\pi$$

ce qui montre la correction de l'algorithme.

On arrive même à majorer la distance entre la fonction de valeur obtenue et celle recherchée, v_π (en quelques sortes donc, l'erreur de l'algorithme):

Proposition 8 *Supposons que l'on ait, au cours de l'itération de* L_π *commencée avec* V_0*, avec* $n \in \mathbb{N}^*$*:*

$$\Big|\Big|L_\pi^n\big(V_0\big) - L_\pi^{n-1}(V_0)\Big|\Big|_\infty \leq \varepsilon$$

avec ε *fixé, alors on aura la majoration suivante pour l'erreur*

$$\Big|\Big|v_\pi - L_\pi^n(V_0)\Big|\Big|_\infty \leq \frac{\gamma\varepsilon}{1-\gamma}$$

Démonstration
Pour simplifier les notations, on note pour tout $n \in \mathbb{N}^*$:

$$u_n = L_\pi^n\big(V_0\big)$$

On a dans un premier temps pour $n \in \mathbb{N}^*$, comme L_π est contractante pour la norme infinie:

$$\begin{aligned}\Big|\Big|u_{n+1} - u_n\Big|\Big|_\infty &= \gamma\Big|\Big|L_\pi\big(u_n\big) - L_\pi\big(u_{n-1}\big)\Big|\Big|_\infty \\ &\leq \gamma\Big|\Big|u_n - u_{n-1}\Big|\Big|_\infty\end{aligned}$$

et par récurrence, on en déduit pour tout $(n, p) \in \mathbb{N}^* \times \mathbb{N}$:

$$\Big|\Big|u_{n+p} - u_{n+p-1}\Big|\Big|_\infty \leq \gamma^p \Big|\Big|u_n - u_{n-1}\Big|\Big|_\infty \quad (*)$$

Soit $(n, p) \in \mathbb{N}^* \times \mathbb{N}$. On a:

$$\begin{aligned}
\Big|\Big|u_{n+p} - u_n\Big|\Big|_\infty &= \Big|\Big| \sum_{k=1}^{p} u_{n+p-k+1} - u_{n+p-k}\Big|\Big|_\infty \quad \text{(somme télescopique)} \\
&\leq \sum_{k=1}^{p} \Big|\Big|u_{n+p-k+1} - u_{n+p-k}\Big|\Big|_\infty \quad \text{(inégalité triangulaire)} \\
&\leq \sum_{k=1}^{p} \Big(\gamma^{p-k+1} \Big|\Big|u_n - u_{n-1}\Big|\Big|_\infty\Big) \quad ((*) \text{ avec } p = p-k+1) \\
&= \Big(\sum_{k=1}^{p} \gamma^k\Big) \Big|\Big|u_n - u_{n-1}\Big|\Big|_\infty \quad \text{(changement d'indice)} \\
&= \gamma \frac{1-\gamma^p}{1-\gamma} \Big|\Big|u_n - u_{n-1}\Big|\Big|_\infty \quad (\gamma \neq 1)
\end{aligned}$$

en faisant tendre p vers $+\infty$ dans l'inégalité précédente, on obtient

$$\Big|\Big|v_\pi - u_n\Big|\Big|_\infty \leq \frac{\gamma}{1-\gamma} \Big|\Big|u_n - u_{n-1}\Big|\Big|_\infty$$

car on sait, d'après la Prop. 7, que l'algorithme d'évaluation de politique converge vers v_π, et que $\gamma \in [0,1[$. Ainsi, si on a $\Big|\Big|u_n - u_{n-1}\Big|\Big|_\infty \leq \varepsilon$, il vient l'inégalité cherchée:

$$\Big|\Big|v_\pi - L_\pi^n(V_0)\Big|\Big|_\infty \leq \frac{\gamma\varepsilon}{1-\gamma}$$

Ainsi, on dispose d'un algorithme permettant de calculer la fonction de valeurs d'une politique donnée.

4 Algorithme de Value Iteration

Toujours dans le but de résoudre un MDP, on cherche la politique optimale, π_*. Cette politique, par définition, prend dans chaque état, et d'une manière déterministe, l'action qui maximise le retour G_t reçu par la suite. (ainsi $\pi(a|s) = 0$ sauf pour une seule action) On va dans un premier temps essayer de déterminer sa fonction de valeurs v_*.

On fixe $t \in \mathbb{N}$, $s \in \mathbb{S}$. Etant donné que π_* est optimale, on a

$$\begin{aligned}
v_*(s) &= \max_a\Big(\mathbb{E}_\pi\big[R_{t+1} + \gamma G_{t+1}|S_t = s\big]\Big) \quad \text{(on récupère la plus haute valeur possible)} \\
&= \max_a\Big(\mathbb{E}_\pi\big[R_{t+1}|S_t = s\big] + \gamma\mathbb{E}_\pi\Big[G_{t+1}|S_t = s\big]\Big) \quad \text{(linéarité de l'espérance)} \\
&= \max_a\Big(\sum_{a\in\mathbb{A}} \pi_*(a|s)R(s,a)\Big) + \max_a\Big(\gamma\sum_{a\in\mathbb{A}} \pi_*(a|s)\sum_{s'\in\mathbb{S}} P(s'|s,a)v_*(s')\Big) \\
&= \max_a\Big(R(s,a)\Big) + \max_a\Big(\gamma\sum_{s'\in\mathbb{S}} P(s'|s,a)v_*(s')\Big) \quad \text{(on cherche l'action qui maximise les valeurs)} \\
&= \max_a\Big(R(s,a) + \gamma\sum_{s'\in\mathbb{S}} P(s'|s,a)v_*(s')\Big)
\end{aligned}$$

On obtient ainsi ce qu'on appelle **l'équation d'optimalité de Bellman**. On en déduit, comme pour l'équation de Bellman, un système de card($\mathbb{S}$) équations que vérifie v_*. Cependant, ce système n'est cette fois-ci pas linéaire à cause de la présence des max. On va le résoudre de manière itérative, avec l'algorithme de Value Iteration.

Cet algorithme, très similaire à l'algorithme d'évaluation de politique, va partir d'une estimation de v_* (quelconque) et va itérativement lui appliquer un certain opérateur H défini par:

$$H : \begin{cases} \mathbb{R}^{\text{card}(\mathbb{S})} & \to & \mathbb{R}^{\text{card}(\mathbb{S})} \\ V & \mapsto & \max_a\, R^a + \gamma T^a V \end{cases}$$

avec R^a le vecteur colonne de longueur card($\mathbb{S}$) qui contient, pour chaque état s, la valeur $R(s,a)$, et T^a la matrice de transition à un pas par rapport à l'action a. Pour $a \in \mathbb{A}$, et si on note $\mathbb{S} = \{s_1,...,s_N\}$, on aura ainsi

$$\forall(i,j) \in [\![1,N]\!]^2 (T^a)_{i,j} = P(s_j|s_i, a)$$

Là encore, il est intéressant de remarquer que $H\big(V\big)$ est une réécriture du membre de droite de l'équation d'optimalité de Bellman.

Proposition 9 *H est contractante pour la norme infinie, i.e. il existe $k \in [0,1[$ tel que*

$$\forall (V_1, V_2) \in (\mathbb{R}^{card(\mathbb{S})})^2, \left|\left|H(V_1) - H(V_2)\right|\right|_\infty \leq k\left|\left|V_1 - V_2\right|\right|_\infty$$

Démonstration
Soit $s \in \mathbb{S}$. On note a_s^* une action (elle n'est pas forcément unique) optimale à prendre dans l'état s. Remarquons que l'on a :

$$a_s^* = \text{argmax}_a R(s,a) + \gamma \sum_{s' \in \mathbb{S}} P(s'|s,a) v_*(s')$$

car on sait que π_* va sélectionner l'action qui maximise l'espérance du retour. On ne cherche pas à exprimer cette action ici.

Soit $(V_1,V_2) \in (\mathbb{R}^{\text{card}(\mathbb{S})})^2$ deux fonctions de valeurs. $H\big(V_1\big)$ et $H\big(V_2\big)$ sont elles aussi des fonctions de valeurs.

Supposons que $H\big(V_1\big)(s) \geq H\big(V_2\big)(s)$ (l'ordre est total sur $\mathbb{R}$). On a, par définition de l'opérateur H:

$$\begin{aligned}
0 &\leq H\big(V_1\big)(s) - H\big(V_2\big)(s) \\
&= R(s,a_s^*) + \gamma \sum_{s' \in \mathbb{S}} P(s'|s,a_s^*) V_1(s') - R(s,a_s^*) - \gamma \sum_{s' \in \mathbb{S}} P(s'|s,a_s^*) V_2(s') \\
&= \gamma \sum_{s' \in \mathbb{S}} P(s'|s,a_s) \Big(V_1(s) - V_2(s) \Big) \quad \text{(linéarité de la somme)} \\
&\leq \gamma \sum_{s' \in \mathbb{S}} \Big(P(s'|s,a) \Big|\Big| V_1 - V_2 \Big|\Big|_\infty \Big) \text{(majoration par la norme infinie)} \\
&= \gamma \Big|\Big| V_1 - V_2 \Big|\Big|_\infty \sum_{s' \in \mathbb{S}} P(s'|s,a) \\
&= \gamma \Big|\Big| V_1 - V_2 \Big|\Big|_\infty \quad \text{(la somme vaut 1)}
\end{aligned}$$

le même calcul pour le cas $H\big(V_2\big)(s) \geq H\big(V_1\big)(s)$ donne la même minoration (étant donné que $||x||_\infty = ||-x||_\infty$).

Ainsi, l'inégalité étant vraie pour tous les états s de $\mathbb{S}$, elle l'est en particulier pour l'état pour lequel la différence entre $H\big(V_1\big)(s)$ et $H\big(V_2\big)(s)$ est maximale, c'est-à-dire qu'on a:

$$\Big|\Big| H\big(V_1\big) - H\big(V_2\big) \Big|\Big|_\infty \leq \gamma \Big|\Big| V_1 - V_2 \Big|\Big|_\infty$$

ce qui montre que H est contractante pour la norme infinie, avec $k = \gamma$, le facteur de remise.

On définit alors l'algorithme de Value Iteration de la manière suivante :

Algorithm 2: Value Iteration

Init $V(s) = 0$ pour chaque $s \in \mathbb{S}$

repeat
  foreach s **do**
    $V(s) \leftarrow \max_a \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \right]$
  end
until *V ne change presque plus*

return $V \approx v_*$

Il s'agit en fait de l'itération de l'opérateur H, à partir d'une fonction de valeur quelconque (ici, la fonction de valeur nulle), c'est-à-dire un élément de $\mathbb{R}^{\text{card}(\mathbb{S})}$.

Proposition 10 *L'algorithme de* Value Iteration *converge vers v_*.*

Démonstration
H est contractante pour la norme infinie sur $\mathbb{R}^{\text{card}(\mathbb{S})}$, qui est un espace vectoriel complet d'après la proposition 2. Ainsi, d'après le théorème du point fixe (proposition 5) avec H, cette application admet un unique point fixe $v \in \mathbb{R}^{\text{card}(\mathbb{S})}$. Ainsi, l'algorithme se termine. De plus, comme v_* vérifie les équations d'optimalité de Bellman, on a :

$$H(v_*) = v_*$$

donc par unicité du point fixe de H, il vient que

$$v = v_*$$

ce qui montre la correction de l'algorithme.

Proposition 11 *Soit V_0 une fonction de valeurs et $\varepsilon > 0$. Si*

$$\left|\left| H^{n+1}(V_0) - H^n(V_0) \right|\right|_\infty \leq \varepsilon$$

alors

$$\left|\left| H^{n+1}(V_0) - v_* \right|\right|_\infty \leq \frac{\varepsilon}{1-\gamma}$$

Démonstration
Même démonstration que la Prop. 8. (on avait simplement utilisé le fait que L_π soit contractante, or H l'est aussi)

On dispose donc d'un algorithme permettant de connaître la fonction de valeur de π_*, avec une précision que l'on sait déterminer (cf Prop. 11). Cependant, nous voulons obtenir π_*. On peut la déduire de v_* en donnant, pour chaque état s, une probabilité de 1 à l'action a qui maximise $R(s,a) + \gamma \sum_{s'} P(s'|s,a)v_*(s')$. Mais nous n'avons accès qu'à une approximation de v_*: il faut donc s'assurer que la fonction de valeur de la politique que l'on va extraire de cette approximation **reste proche** de v_*.

Proposition 12 *Soit V_0 une fonction de valeurs et $\varepsilon > 0$. Si*

$$\left|\left|H^{n+1}(V_0) - H^n(V_0)\right|\right|_\infty \leq \varepsilon$$

alors, en notant π_n la politique définie par

$$\forall s \in \mathbb{S},\ \pi_n(s) = argmax_a\Big(R(s,a) + \gamma \sum_{s' \in \mathbb{S}} P(s'|s,a)H^n(V_0)\Big)$$

on aura

$$\left|\left|v_{\pi_n} - v_*\right|\right|_\infty \leq \frac{2\gamma\varepsilon}{1-\gamma}$$

Démonstration
On a:

$$\begin{aligned}
\left|\left|v_{\pi_n} - v_*\right|\right|_\infty &= \left|\left|v_{\pi_n} - H^n(V_0) + H^n(V_0) - v_*\right|\right|_\infty \\
&\leq \left|\left|v_{\pi_n} - H^n(V_0)\right|\right|_\infty + \left|\left|H^n(V_0) - v_*\right|\right|_\infty \\
&\leq \frac{\gamma\varepsilon}{1-\gamma} + \frac{\gamma\varepsilon}{1-\gamma} \quad \text{(Prop. 8 et 11)} \\
&= \frac{2\gamma\varepsilon}{1-\gamma}
\end{aligned}$$

On peut utiliser la proposition 8 car on a utilisé, pour créer π_n, l'opérateur de l'algorithme d'Evaluation de politique L_{π_n}. L'utilisation de la proposition 11 nous est directement permise par l'inégalité supposée.

Proposition 13 *La complexité temporelle de* Value Iteration *est en* $\Theta\Big(n|\mathbb{S}|^2|\mathbb{A}|\Big)$, *en notant* n *le nombre d'itérations effectuées,* $|\mathbb{S}| = card(\mathbb{S})$ *et* $|\mathbb{A}| = card(\mathbb{A})$.

Démonstration
Calculons dans un premier temps la complexité temporelle pour une seule itération.

L'algorithme de Value Iteration peut s'écrire:

```
value_iteration:
  pour chaque s ∈ 𝕊:
       pour chaque a ∈ 𝔸:
            pour chaque s' ∈ 𝕊:
                 mise à jour de V(s)
```

ce qui revient à parcourir tous les triplets $(s,a,s') \in \mathbb{S} \times \mathbb{A} \times \mathbb{S}$ de la manière suivante :

```
value_iteration:
  pour chaque (s,a,s') ∈ 𝕊 × 𝔸 × 𝕊:
       mise à jour de V(s)
```

avec l'étape `mise à jour de` $V(s)$ qui se fait en temps constant. On fait donc $\text{card}(\mathbb{S} \times \mathbb{A} \times \mathbb{S}) = |\mathbb{S}|^2|\mathbb{A}|$ passages dans la boucle, et chaque passage est en temps constant. Ainsi, la complexité d'une itération est

$$\Theta\Big(|\mathbb{S}|^2\mathbb{A}\Big)$$

et donc, si on effectue n itérations, chaque itération est en temps constant **par rapport** à n, d'où la complexité en

$$\Theta\Big(n|\mathbb{S}|^2|\mathbb{A}|\Big)$$

pour l'algorithme de Value Iteration.

5 Application : l'environnement Frozen Lake

5.1 Implémentation Python

On a l'implémentation suivante de l'algorithme de Value Iteration, en Python:

```
def q_value_pour_s_a(env, V, s, a, gamma):
    q = 0

    for (p_sPrime, sPrime, r_ss_a, done) in env.P[s][a]:
        q += p_sPrime * (r_ss_a + gamma * V[sPrime])

    return q

def value_iteration(gamma, nS, nA, epsilon):
    V = np.zeros([nS, 1])
    n=0
    delta = 1000

    while delta >= epsilon:
        n += 1
        delta = 0

        for s in range(nS):
            q_s = np.zeros([nA, 1])

            for a in range(nA):
                q_s[a] = q_value_pour_s_a(env, V, s, a, gamma)

            V_nouvelle = np.max(q_s)
            delta = max(delta, np.abs(V_nouvelle - V[s]))
            V[s] = V_nouvelle
    return (V,n)

def improve_policy(env, pi, V, gamma):
    for s in range(env.nS):
        q_s = np.zeros([env.nA, 1])

        for a in range(env.nA):
            q_s[a] = q_value_pour_s_a(env, V, s, a, gamma)

        best_a = np.argmax(q_s)
        pi[s] = np.eye(env.nA)[best_a]
    return pi
```

la fonction `q_value_pour_s_a` permet de calculer, pour s et a donnés, la valeur

$$R(s,a) + \gamma \sum_{s' \in \mathbb{S}} P(s'|s,a)V(s')$$

ainsi, la fonction `value_iteration` répète essentiellement une boucle de mises à jour dans laquelle, à chaque itération, elle calcule cette valeur pour chaque état et pour chaque action, et choisi alors la valeur maximale (selon l'action), à l'aide de `np.max`.

Notons tout de même qu'ici, la fonction `q_value_pour_s_a` calcule en réalité la valeur :

$$\sum_{s' \in \mathbb{S}} R(s,a,s') + \gamma P(s'|s,a) V(s')$$

cela est du à une définition légèrement différente d'un MDP : dans ce cas, la fonction de récompense considérée ne dépend pas seulement de s et a, mais aussi de s', c'est-à-dire l'état atteint au même pas de temps que la récompense est reçue. Cela modifie ainsi légèrement les équations de Bellman, mais n'a aucun impact sur la théorie des algorithmes vus.

La fonction `improve_policy` permet d'extraire de notre approximation de v_* une politique : avec la Prop. 12, on sait que cette politique est proche de π_*.

5.2 Environnement FrozenLake

Nous allons employer l'algorithme de Value Iteration sur l'environnement FrozenLake. FrozenLake est un environnement assez basique, souvent utilisé en tant que démonstration dans l'apprentissage par renforcement. Il est proposé et implémenté en Python par OpenAI, une entreprise pionnière dans le domaine de l'apprentissage par renforcement.

Cet environnement est une sorte de labyrinthe avancé : l'agent se déplace sur des cases, il y en a 16 dans la version de base. L'agent peut aller en haut, en bas, à gauche ou à droite (lorsque cela est permis : si un mur l'en empêche, il restera alors à sa position).

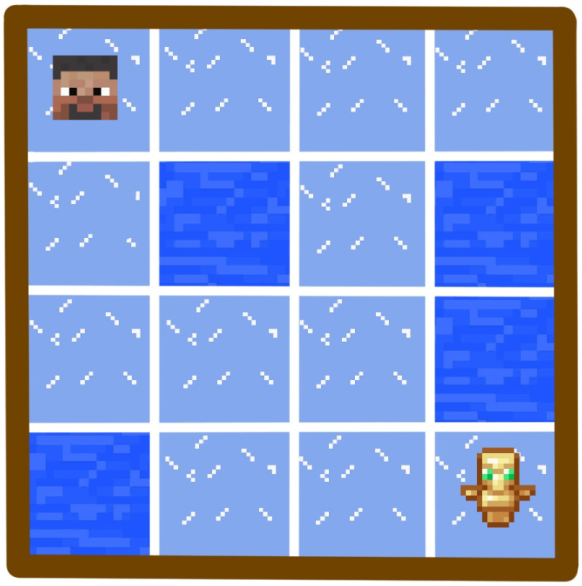

Fig. 3 – *Représentation graphique de FrozenLake*

Le but pour l'agent est de visiter la case située en bas à droite, en commençant sur la case en haut à gauche. Pour complexifier la tâche, on suppose que l'agent se déplace sur une sorte de lac gelé. Ainsi, certaines cases ne sont pas de la glace mais de l'eau (cases de

couleur bleu foncé sur la Figure 3, si l'agent atterri sur cette case, l'interaction en cours se termine avec +0 de récompense. En revanche, si l'agent parvient à naviguer jusqu'à la dernière case, alors il reçoit une récompense de +1, et l'interaction s'arrête alors.

Pour rendre la tâche plus compliquée encore, on suppose que les actions de l'agent ne sont pas certaines. On modélise en quelques sortes le fait que l'agent peut glisser lorsqu'il se déplace sur la glace. Ainsi, lorsqu'il choisit une direction, il y a seulement une probabilité de $\frac{1}{3}$ pour que l'agent se rende dans la case désirée. Il a ainsi une probabilité de $\frac{1}{3}$ de se déplacer (sans le vouloir donc) à gauche, et $\frac{1}{3}$ de se déplacer à droite :

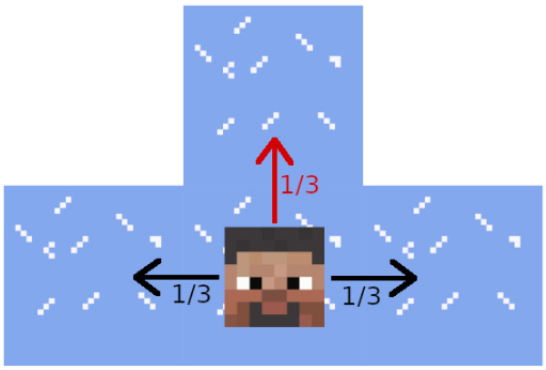

FIG. 4 – *La glace, ça glisse!*

(si jamais un mur se trouve à droite ou à gauche de l'agent, l'agent a alors une certaine probabilité de faire du sur-place)

Il s'agit maintenant de définir un MDP à partir de la description de l'environnement précédente. Les états, au nombre de 16, correspondront aux 16 cases sur lesquelles l'agent peut se rendre. Les actions seront au nombre de 4 : gauche, bas, droite, haut. Les récompenses seront toutes nulles, sauf dans le cas où on atterri dans l'état correspondant à la case située en bas à droite. Enfin, les probabilités de transitions seront celles décrites plus haut. Nous travaillerons avec $\gamma = 0{,}99$.

On y applique alors l'algorithme de VALUE ITERATION implémenté précédemment. Après `77` itérations, l'algorithme s'arrête (on a choisi `epsilon=1e-5`). La fonction de valeur sur laquelle il a convergé est la suivante :

FIG. 5 – v_* - *la couleur de l'état représente sa valeur. Jaune = proche de* 1*, violette = proche de* 0

On remarque, sans surprise, que les états proches de l'état qui donne une récompense ont leur valeur proche de 1. Les états correspondant à de l'eau ont une valeur de 0: une fois arrivé dans cet état, le jeu s'arrête immédiatement et aucune récompense n'est reçue. On remarque qu'il en va de même pour l'état final.
Voici alors la politique π_* extraite de v_* :

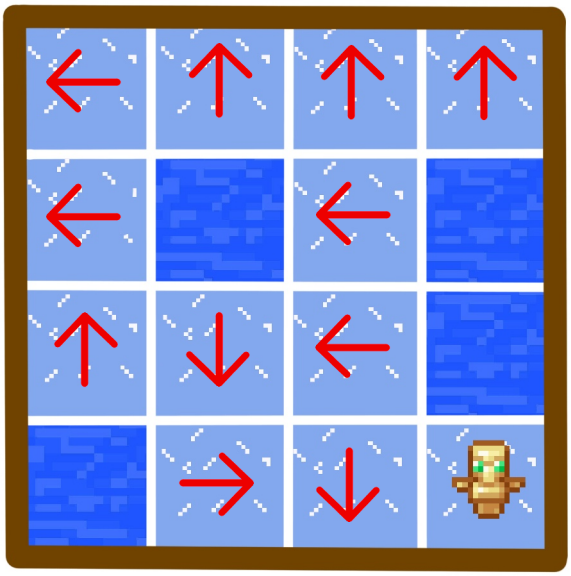

FIG. 6 – *La politique optimale sur laquelle a convergé* VALUE ITERATION

Cette politique est optimale, bien que cela n'apparaisse pas à première vue. Il peut paraître contre-intuitif, par exemple, de vouloir aller directement dans l'eau à l'état 6 (en numérotant les états de 0 à 15, de en haut à gauche jusqu'à en bas à droite), l'agent, en faisant cela, n'a en réalité que 1 chance sur 3 d'aller dans l'eau. Si il avait choisit l'action d'aller en bas par exemple, du aux dynamiques de l'environnement, il aurait 2 chances sur 3 d'aller dans l'eau.

Cet exemple ne pénalise pas l'agent pour sa lenteur. Si on voulait trouver une stratégie qui, quitte à prendre un peu plus de risque, soit plus rapide, il suffirait de modifier la fonction de récompense : par exemple, on pourrait infliger une récompense de -0.1 à chaque pas de temps (la magnitude de cette récompense traduit le compromis temps-risque).

5.3 FrozenLake, avec card($\mathbb{S}$) variable

L'environnement FrozenLake peut être agrandi. On choisit ici d'avoir une grille de 16 cases par 16 cases, i.e. $\text{card}(\mathbb{S}) = 256$.

Après 202 itérations, on a le résultat suivant :

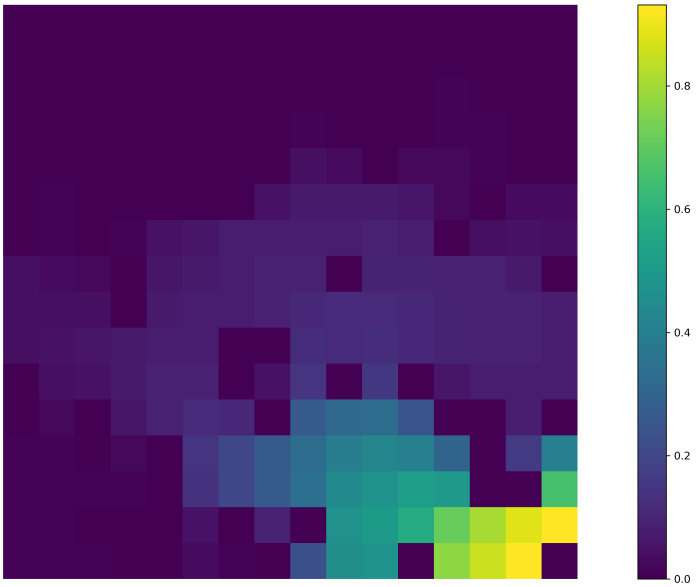

FIG. 7 – v_* - *Valeurs des 256 états.*

D'autres valeurs ont été essayées pour FrozenLake. En voici les résultats :

$n = \text{card}(\mathbb{S})$	temps	nombre d'itérations
16	0,1	77
100	1,4	202
1000	17,8	274
10000	123,4	208

On remarque que la complexité observée ici semble être linéaire pour le nombre d'états (alors que d'après Prop. 13, elle devrait être quadratique pour le nombre d'états). En fait, elle l'est réellement mais c'est la façon dont est implémenté l'environnement qui en est responsable. En effet, pour chaque état s, l'environnement nous donne seulement, pour les états s' suivants, **ceux qui ont une probabilité non nulle d'intervenir**. On a donc beaucoup moins d'états à regarder (seulement les états voisins à l'état actuel : un nombre constant peu importe le nombre d'états total), ce qui fait que la complexité est ici linéaire pour le nombre d'états.

6 Conclusion et discussions

Nous avons ainsi résolu le problème posé dans la partie, à savoir résoudre un MDP. Ainsi, pour un environnement donné que l'on transcrit en MDP, on sait construire un enchaînement optimal d'actions pour maximizer les récompenses reçues.

La construction des algorithmes qui a été faite à toutefois été menée suite à une hypothèse majeure faite en début de partie : la connaissance des probabilités de transition entre chaque état ainsi que la connaissance des récompenses reçues. Il est clair que cette hypothèse est un obstacle important à la mise en place de l'algorithme de VALUE ITERATION sur la majorité des problèmes réels : pour un environnement possédant n_s états et n_a actions, il faudrait connaître $n_s n_a$ récompenses ainsi que $n_s^2 n_a$ probabilités. A moins de connaître parfaitement le phénomène étudié, il peut être difficile de calculer chacune des ses récompenses et probabilités : par exemple, si l'on considère un jeu dans lequel l'algorithme doit affronter des joueurs extérieurs (ou même, d'autres algorithmes, qui eux aussi évoluent), il est difficile de calculer les transitions de l'environnement.

De la même façon, il a été supposé dans ce document que nous travaillons avec des espaces d'états finis - en particulier donc, les ensembles $\mathbb{S}$ et $\mathbb{A}$ doivent être finis. Là aussi, il s'agit d'un gros frein à la mise en place de l'algorithme de VALUE ITERATION sur des problèmes réels.

Par exemple, si l'on souhaite faire apprendre à un algorithme à faire tenir un bâton en équilibre en déplaçant un chariot (rectangle noir sur la figure 8), alors les deux problèmes précédemment soulevés vont rendre l'implémentation de VALUE ITERATION impossible ici.

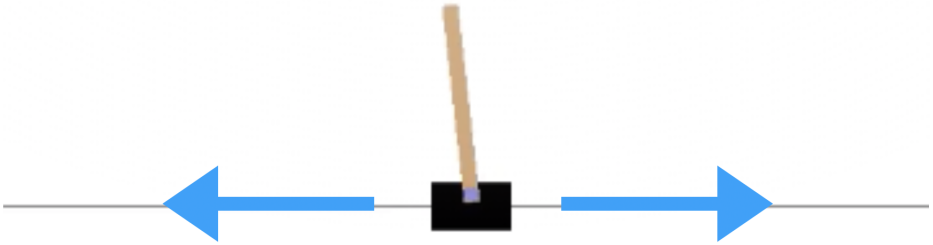

FIG. 8 – *L'environnement* CARTPOLE*, également proposé par* OPENAI

En effet, nous devrons dans un état donner la description de l'état du chariot et du bâton. Or, ceux-ci sont caractérisés par leur position ou bien même leur vitesse, qui sont des réels : l'ensemble des états de notre MDP serait alors infini. Peut-être pourrait on le discrétiser, c'est-à-dire créer des groupes d'états que l'on considère comme assez proches, et considérer comme l'ensemble des états ces différents groupes d'états. Mais, en plus de rendre notre solution inexacte quant au problème original, cela se répercuterait sur le premier problème évoqué : pour avoir une solution assez bonne, il faudrait créer énormément de petits paquets d'états, et donc card($\mathbb{S}$) serait très grand. Il faudrait alors, pour chacun de ces états, estimer toutes les probabilités requises, ainsi que les récompenses !

Une stratégie très utilisée pour faire face au premier problème, celui qui est de connaître les probabilités de transitions et récompenses, est d'utiliser la loi des grands nombres dans l'équation de Bellman. Au lieu de prendre $R(s,a)$, on utilise la récompense reçue, et au lieu de considérer tous les états possibles suivants, on considère seulement celui qu'on a obtenu lors de notre interaction avec l'environnement. Ainsi, dans ce cas, l'agent interagit **réellement** [1] avec l'environnement pour y estimer les bonnes et mauvaises actions à prendre pour maximiser les récompenses : certains parlent alors d'*apprentissage*.

1. dans le cas de VALUE ITERATION, l'algorithme n'interagit pas avec l'environnement : il ne fait que regarder ses probabilités de transitions et ses récompenses.

Références

- Ressources générales
  1. Sabin Lessard. *Processus stochastiques*. 2014
- Apprentissage par renforcement
  1. Richard S. Sutton et Andrew G. Barto. *Reinforcement Learning : An Introduction*. 2017
  2. Pascal Poupart. *CS885, University of Waterloo*. 2018
  3. Balaraman Ravindran, NPTEL. *Reinforcement Learning*. 2018
  4. OpenAI Gym (`https://gym.openai.com/`)
  5. David Silver. *Introduction to Reinforcement Learning, DeepMind*. 2015