ÉCOLE NATIONALE SUPÉRIEURE DE TECHNIQUES AVANCÉES

ROB311 : Apprentissage pour la robotique

Parcours robotique

Année universitaire : 2020/2021

RL : Algorithmes "Value Iteration" et "Policy Iteration"

Auteurs :

M. AHMED YASSINE HAMMAMI

MME. HANIN HAMDI

Objectifs

L'objectif de ce TP2 est d'exécuter et implémenter les deux algorithmes d'apprentissage par renforcement (RL) :

- Value Iteration
- Policy Iteration

Question 1

Les actions possibles pour chaque état sont :

$$\pi(s_0) = \{a_1, a_2\}$$
$$\pi(s_1) = \{a_0\}$$
$$\pi(s_2) = \{a_0\}$$
$$\pi(s_3) = \{a_0\}$$

Question 2

Pour répondre à cette question et aux deux questions qui suivent, on suit les étapes de l'algorithme "policy iteration" suivantes :

1. Initialiser Q(s,a) arbitrairement, pour tout état s et pour toute action a.
2. Répéter pour chaque épisode :
   (a) Trouver pour chaque action et pour chaque état de la matrice π et calculer la fonction de valeur à partir de l'équation de Bellman :
   $$V^*(S_t) = R(S_t) + max_a \gamma \sum T(S_t, a, S_{t+1}) V^*(S_{t+1})$$
   (b) Pour chaque état, choisir l'action optimale et mettre à jour π selon :
   $$\pi(S_t) = arg\ max_a \sum T(S_t, a, S_{t+1}) V^*(S_{t+1})$$
   (c) Répéter ces deux étapes pour chaque état jusqu'à convergence. Dans notre cas, on considère que l'algorithme converge si ma matrice π ne change pas.

Alors, initialement $\pi = \{a_1, a_0, a_0, a_0\}$.

Itération 1 : On suppose que les valeurs initiales de $V(s)$ pour tous les états sont nulles. Alors : $V^*(S) = R(S)$. Ainsi, on a :

$$V(s_0) = 0$$
$$V(s_1) = 0$$
$$V(s_2) = 1$$
$$V(s_3) = 10$$

L'amélioration de π est : Pour tous les états s_1, s_2, et s_3, aucune amélioration possible pour π car on a toujours une seule action possible a_0.

Pour l'état s_0 : deux actions possibles a_1 et a_2 :

$$a_1 : \gamma \sum T(S_0, a_1, S_1) V^*(S_1) = 0$$
$$a_2 : \gamma \sum T(S_0, a_2, S_2) V^*(S_2) = \gamma$$

Le max c'est γ donc $\pi^*(s0) = a_2$.
Conclusion : $\pi^* = [a_2, a_0, a_0, a_0]$

Itération 2 :

$$V^*(s_0) = R(s_0) + \gamma max\{V^*(s_1), V^*(s_2)\} = \gamma$$

$$V^*(s_1) = R(s_1) + \gamma[T(s_1, a_0, s_1)V^*(s_1) + T(s_1, a_0, s_3)V^*(s_3)] = \gamma[(1-x)V^*(s_1) + xV^*(s_3) = 10\gamma x$$

$$V^*(s_2) = R(s_2) + \gamma[T(s_2, a_0, s_0)V^*(s_0) + T(s_2, a_0, s_3)V^*(s_3)] = 1 + \gamma[(1-y)V^*(s_0) + yV^*(s_3) = 1 + 10\gamma y$$

$$V^*(s_3) = R(s_3) + \gamma V^*(s_0) = 10$$

L'amélioration de π pour l'état s_0 :

$$a_1 : \gamma \sum T(S_0, a_1, S_1)V^*(S_1) = 10\gamma x$$
$$a_2 : \gamma \sum T(S_0, a_2, S_2)V^*(S_2) = 1 + 10\gamma y$$

On s'arrête à ce stade car on a besoin de plus d'information concernant x, y et γ pour pouvoir décider de l'amélioration.

Question 3 :

soient y et γ deux valeurs quelconques de $[0, 1]$ et $]0, 1]$ (on élimine le cas où $\gamma = 0$) :

$\pi^*(s_0) = a_2 \iff 1 + 10\gamma y > 10\gamma x$
$\iff x < y + \frac{1}{10\gamma}$.

Ainsi, on peut choisir $x = 0$.

Question 4 :

Soient x et γ deux valeurs quelconques de $[0, 1]$ et $]0, 1]$ (on élimine le cas où $\gamma = 0$) :

Supposons qu'il existe y dans [0,1] tel que x $\in [0, 1]$ *et* $\gamma \in]0, 1]$: $\pi^*(s_0) = a_1$
$\iff 1 + 10\gamma y < 10\gamma x$
$\iff y < x - \frac{1}{10\gamma}$. Par exemple, prenons $x = 0$
$\iff y < -\frac{1}{10\gamma} < 0$ ce qui est absurde. Donc cette proposition est fausse.

Question 5 :

Prenons $x = y = 0.25$ et $\gamma = 0.9$. On a alors : $10\gamma x = 2.25$ et $10\gamma y + 1 = 3.25$. Ainsi, l'action optimale à prendre à l'état s_0 est a_2. On a alors $V^* = [0.9, 2.25, 3.25, 10]$ et $\pi^* = [a_2, a_0, a_0, a_0]$. π^* est identique à sa valeur calculée à l'itération précédente donc l'algorithme s'arrête ici.

Pour la suite du TP2, on implémente l'algorithme "Value Iteration" avec un code Python. On obtient les valeurs suivantes de la matrice de l'Utilité optimale de chaque état et la matrice π^* optimale :

Tel que les valeurs 1 et 0 dans π font référence respectivement aux actions a_1 et a_0. De plus chaque colonne de UTILITY et π correspond à un état s.

```
OPTIMAL UTILITY

[14.18376816 15.7599503  15.69602736 22.76520424]
```

FIGURE 1 – La matrice de l'utilité optimale

```
OPTIMAL POLICY

[1. 0. 0. 0.]
```

FIGURE 2 – π optimale

Remarque : On constate une différence entre le résultat théorique et le résultat pratique quant à la matrice π optimale. Ceci est dû à la façon par laquelle on décide d'arrêter l'algorithme. En fait, pour le calcul théorique on décide d'arrêter le processus dès qu'on obtient deux fois de suite la même matrice π, c'est-à-dire lorsqu'on ne trouve pas une amélioration, c'est pour cela qu'on obtient une action a_2 optimale à partir de l'état s_0. En contrepartie, pour le calcul par le code python, on décide d'arrêter le calcul dès qu'on obtient une matrice d'utilité U qui donne une erreur quadratique moyenne par rapport à l'ancienne U calculée dans l'itération précédente qui est inférieure à 10^{-4}. Et du coup, on se base sur la convergence de U au lieu de π et c'est d'ailleurs la diffrence entre les deux algorithmes de RL : Value Iteration et Policy Iteration.

La figure suivante illustre ce qu'on vient de dire :

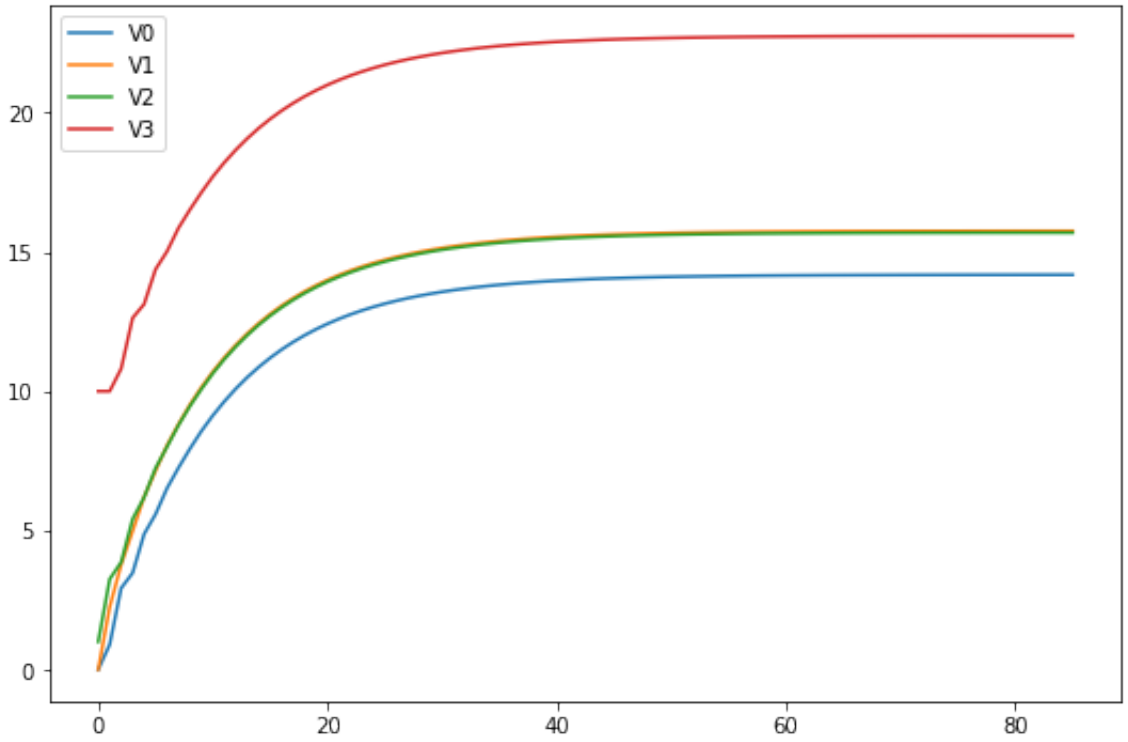

FIGURE 3 – Les améliorations calculée à chaque itération et pour les différents états

On remarque qu'aux 10 premières itérations, V_1 (l'amélioration qu'on obtient si l'on réalise l'action a_1 à partir de l'état s_0) est inférieure à V_2 (l'amélioration qu'on obtient si

l'on réalise l'action a_2 à partir de l'état s_0). Par contre, à partir de la dixième itération, l'algorithme qui calcule l'utilité commence à converger encore plus et V_1 devient à peine supérieure à V_2 (une différence de l'ordre de 10^{-1}). Par contre, pour le calcul théorique, on s'est contenté des deux premières itérations au bout desquelles V_2 est supérieure à V_1.