Apprentissage par renforcement (3)

Bruno Bouzy
12 octobre 2010

Ce document est la troisième partie du cours « apprentissage par renforcement ». La seconde partie portait sur la programmation dynamique, méthode pionnière à la base des méthodes d'apprentissage par renforcement présentées ici. Cette partie présente successivement la méthode **Rmax**, la méthode de **Monte-Carlo**, les méthodes des **Différences Temporelles** (DT ou TD), **Q-learning** et **Sarsa**. Chacune de ces méthodes sont illustrées avec le même exemple: celui du GridWorld présenté dans la partie 1 du cours.

Rmax

Introduction

Dans cette partie, nous présentons **Rmax**, un algorithme d'apprentissage par renforcement très simple utilisant un modèle de l'environnement [Brafman & Tennenholtz 2002]. Dans Rmax, l'agent maintient un modèle de l'environnement (c'est-à-dire les probabilités de transitions $P^a_{ss'}$ et les retours $R^a_{ss'}$ du PDM). Ce modèle est inexact au début mais se raffine au fur et à mesure. L'agent agit toujours optimalement par rapport à ce modèle en utilisant l'algorithme "Value Iteration" (VI) pour calculer V et Q. Théoriquement VI est appelé à chaque fois que ce modèle est modifié. Ainsi, V et Q sont les fonctions de valeurs optimales par rapport au modèle de l'environnement. Le modèle de l'environnement est initialisé de manière optimiste: tous les retours $R^a_{ss'}$ du PDM sont initalisés au retour maximal. Pendant l'exécution de Rmax, les retours observés viennent remplacer les retours du modèle.

Propriétés

Rmax est une amélioration de Value Iteration car il est « **online** ». Value Iteration est un algorithme « **off-line** » au sens ou le calcul de V et Q est fait une fois pour toute avant utilisation de V et Q par l'agent. Dans Rmax, l'agent commence à agir dans l'environnement et la calcule de V et Q se fait peu à peu au fur et à mesure que l'agent agit dans l'environnement.

Rmax résout le **dilemme exploitation - exploration**. Un problème à résoudre pour un agent apprenant par renforcement est le dilemme entre explorer des actions inconnues et exploiter des actions connues. A un instant donné, lorsque l'espace des états n'est pas complètement exploré, les retours $R^a_{ss'}$ des transitions allant vers des états non explorés valent Rmax. Ainsi VI calcule une politique optimale pour aller vers ces états inconnus à grands retours, donc Rmax explore. De plus, Rmax explore optimalement. On dit que le dilemme exploitation – exploration est résolu **implicitement** par Rmax. En effet, Rmax marche toujours de la même

manière: optimalement par rapport au modèle de l'environnement et n'a pas de mécanisme explicite pour dire je suis en train d'explorer ou d'exploiter.

Rmax suit le biais de l'**optimisme dans l'incertain**. Par défaut, Rmax va vers les états qu'il ne connaît pas en supposant qu'ils sont bons.

Rmax est **simple**. Cf code au paragraphe suivant.

Rmax est général: il s'applique non seulement à des PDM mais aussi à des **jeux stochastiques**. (Un jeu stochastique est un PDM multi-agent: plusieurs agents interagissent dans le graphe d'états. L'état suivant est déterminé par l'action jointe: combinaison des actions des agents.)

Algorithme

Initialisation:
Entrée: longueur de la politique = T
Construction du modèle M' constitué des N états réels G_1, G_2, ..., G_N, plus un état fictif G_0.
Chaque état possède:
- Une valeur d'état V.
- Une matrice d'actions jointes contenant dans chaque cellule:
  - Le retour observé initialisé avec R_{max}.
  - La liste des états suivants rencontrés avec pour chaque état suivant
    - le nombre de fois où la transition a été effectuée.
  - (liste initialisée avec G_0, avec une unique transition fictive effectuée).
  - Un attribut booléen « connu » ou pas, initialisé a faux.
  - Une valeur d'action jointe Q.

Exécution
Répéter
- (C) Calcule la politique optimale de longueur T depuis l'état courant,
- Exécute cette politique pendant T pas de temps.
- Après chaque action jointe et transition vers un état suivant,
  - mettre à jour le retour observé
  - incrémenter le compteur de cette transition.
  - Si le compteur dépasse un seuil, alors
    - l'action jointe est connue
    - aller en (C)

Convergence théorique

Dans [Brafman & Tennenholtz 2002], il est prouvé que Rmax converge vers la politique optimale.

Monte-Carlo

Introduction

Dans cette partie, nous voyons comment associer l'idée de la programmation dynamique avec l'idée de Monte-Carlo (MC). Le terme Monte-Carlo consiste à effectuer des simulations au hasard, à retenir les résultats des simulations et à calculer des *moyennes* de résultats. La stratégie du Monte Carlo est adaptée lorsque l'on ne connaît pas le modèle du domaine (les probabilités de transitions $P^a_{ss'}$ et les retours $R^a_{ss'}$ du PDM). C'est une stratégie simple. Il faut que les tâches de l'agent soient décomposées en épisodes, et un épisode correspondra à une simulation. Dans ce chapitre, à partir d'une politique π, nous regardons comment obtenir une estimation de la fonction de valeur d'états V^π, puis comment estimer une fonction de valeur d'actions Q^π, puis comment améliorer une politique (ce qu'on appelle le « contrôle Monte Carlo »).

Evaluation Monte Carlo de politique

Dans cette partie, on suppose, la politique π donnée et on cherche à l'évaluer avec la fonction de valeur d'état V^π. La stratégie consiste à lancer des épisodes dans lequel l'agent utilise la politique π. Dans un épisode, pour chaque état, on enregistre le résultat de l'épisode et on dit que V(s) est la moyenne des résultats enregistrés. V(s) converge vers $V^\pi(s)$. La loi des grands nombres dit que l'écart-type autour de la moyenne décroît en $1/n^{1/2}$ où n est le nombre de résultats moyennés. le pseudo-code de l'évaluation MC d'une politique est donnée par la figure 19.

Initialize:
$\pi \leftarrow$ policy to be evaluated
$V \leftarrow$ an arbitrary state-value function
$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:
(a) Generate an episode using π
(b) For each state s appearing in the episode:
$R \leftarrow$ return following the first occurrence of s
Append R to $Returns(s)$
$V(s) \leftarrow \text{average}(Returns(s))$

Figure 19: Méthode de Monte Carlo pour estimer V^π.

Le but d'un diagramme de back-up est de montrer en haut l'état à mettre à jour et en dessous les transitions et états suivants contribuant au calcul de la valeur de l'état du haut. Le diagramme de back-up de la méthode MC pour évaluer une politique est donné par la figure 20. On démarre d'un état et on effectue une série de transitions conduisant à un état final où le

résultat est la somme des récompenses reçues pendant l'épisode. Ce résultat est enregistré dans l'état de départ pour servir au calcul de la moyenne.

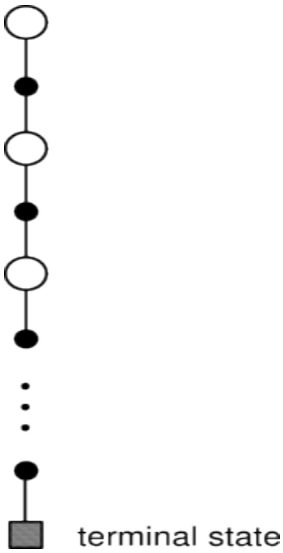

Figure 20: le diagramme de back-up de l'estimation Monte Carlo de V^{π}.

Dans cette méthode, la valeur d'un état est indépendante de la valeur d'un autre état voisin. En ce sens, on dit que la méthode de Monte-Carlo ne « bootstrap » pas comme le fait la PD. Par ailleurs on voit que le coût de cette méthode ne dépend que de la longueur des épisodes. Et que l'on peut estimer les valeurs d'états d'un sous-ensemble de l'espace des états, si seul ce sous-ensemble nous intéresse, et ignorer complètement les autres états.

Estimation Monte Carlo de valeurs d'action

On peut également estimer les valeurs d'actions. Le but est d'estimer $Q^{\pi}(s, a)$ pour chaque état s et action a. La méthode fonctionne pareil que la précédente pour estimer $V^{\pi}(s)$.

Exploration et exploitation

Si la politique et l'environnement sont déterministes, il est possible que des états importants ne soient jamais visités. Pour que les deux précédentes méthodes marchent avec une politique déterministe, il faut absolument maintenir un bon niveau d'exploration au départ de l'état concerné. Pour cet état, il faut que chaque action soit choisie un nombre infini de fois. Cela s'appelle l'hypothèse des « démarrages exploratoires »[1]. Sans cette hypothèse et avec une politique et un environnement déterministes, les méthodes précédentes ne marchent pas. Si la politique est stochastique, l'hypothèse ES n'est pas nécessaire.

[1]hypothèse des « exploring starts » (ES) en anglais.

Contrôle Monte Carlo

Le contrôle Monte Carlo correspond à un processus alternant une amélioration de politique et à une évaluation de politique comme le montre la figure 21, c'est-à-dire à un exemple d'IGP mentionnée au chapitre précédent.

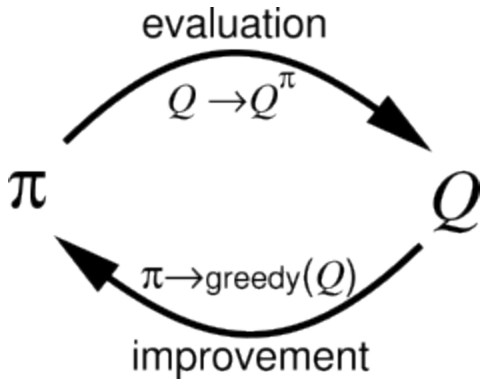

Figure 21: Evaluation et amélioration de valeurs d'action.

La figure 22 donne un exemple d'algorithme de contrôle MC avec politique déterministe et hypothèse ES.

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
$Q(s,a) \leftarrow$ arbitrary
$\pi(s) \leftarrow$ arbitrary
$Returns(s,a) \leftarrow$ empty list

Repeat forever:
(a) Generate an episode using exploring starts and π
(b) For each pair s,a appearing in the episode:
$R \leftarrow$ return following the first occurrence of s,a
Append R to $Returns(s,a)$
$Q(s,a) \leftarrow \text{average}(Returns(s,a))$
(c) For each s in the episode:
$\pi(s) \leftarrow \arg\max_a Q(s,a)$

Figure 22: ES Monte Carlo: un algorithme de contrôle Monte Carlo.

On peut envisager de ne pas faire l'hypothèse des « exploring starts » (ES) mais alors il faut rendre la politique stochastique. Il y aurait deux méthodes à discuter: les méthodes avec politique en ligne[2] et les méthodes avec politiques hors ligne[3]. Mais ce n'est pas l'objet de ce cours.

[2]« on-policy » en anglais.
[3]« off-policy » en anglais.

Monte-Carlo sur l'exemple du Grid World:

Voici une sortie de l'exemple du GridWorld pour le contrôle Monte Carlo:

```
Monte Carlo Control: debut:
  <^>v    <^>v    <^>v    <^>v    <^>v
  <^>v    <^>v    <^>v    <^>v    <^>v
  <^>v    <^>v    <^>v    <^>v    <^>v
  <^>v    <^>v    <^>v    <^>v    <^>v
  <^>v    <^>v    <^>v    <^>v    <^>v
Monte Carlo Control: iteration 1:
  6.14   9.86   5.81   6.06   3.89
  3.80   4.33   3.41   3.10   2.58
  2.20   2.19   1.91   1.73   1.61
  1.35   1.28   1.16   1.08   1.06
  1.04   0.95   0.89   0.84   0.85
    >     <^>v    <       <^>v    <
   ^       ^       ^       ^       ^
   ^       ^       ^       ^       ^
   ^       ^       ^       ^       ^
   ^       ^       ^       <^      ^
Monte Carlo Control: iteration 2:
 19.66  21.87  19.66  16.51  14.88
 17.72  19.66  17.72  14.88  13.37
 15.96  17.72  15.96  13.37  12.01
 14.36  15.96  14.36  12.01  10.82
 12.87  14.36  12.87  11.20   9.73
    >     <^>v    <       <^>v    <
   ^>      ^      <^      <       <^
   ^>      ^      <^      <       <^
   ^>      ^      <^      <       <^
   ^>      ^      <^      <       <
Monte Carlo Control: iteration 3:
 19.66  21.87  19.66  17.37  15.63
 17.72  19.66  17.72  15.96  14.22
 15.96  17.72  15.96  14.36  12.82
 14.36  15.96  14.36  12.87  11.55
 12.87  14.36  12.87  11.57  10.43
    >     <^>v    <       <^>v    <
   ^>      ^      <^      <       <
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
Monte Carlo Control: iteration 4:
 19.66  21.87  19.66  17.37  15.63
 17.72  19.66  17.72  15.96  14.36
 15.96  17.72  15.96  14.36  12.87
 14.36  15.96  14.36  12.87  11.57
 12.87  14.36  12.87  11.57  10.43
    >     <^>v    <       <^>v    <
   ^>      ^      <^      <       <
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
Monte Carlo Control, fin.
```

TD-learning

introduction

Si l'on devait identifier une idée centrale et nouvelle de l'AR, c'est sans doute la méthode des Différences Temporelles (DT[4]). L'apprentissage TD est une association des idées de la PD et de MC. Comme MC, TD apprend à partir de l'expérience directe de l'agent, sans modèle de la dynamique de l'environnement. Comme PD, TD met à jour les états en fonction de leurs voisins, sans dépendre des états finals. En ce sens, TD « bootstrap ». Comme dans les autres chapitres, nous commençons par le cas de l'évaluation d'une politique, ce que nous appelons la prédiction TD, puis nous décrirons deux méthodes de contrôle, Sarsa et Q-learning, recherchant à optimiser des fonctions de valeurs d'action.

Prédiction TD

Dans le chapitre précédent sur MC, on aurait pu écrire la règle de mise à jour de la manière suivante:

$$V(s_t) \leftarrow V(s_t) + \alpha [R_t - V(s_t)] \qquad (30)$$

Nous appelons cette mise à jour celle de *constant-α MC*. Alors que MC doit attendre la fin d'un épisode pour mettre à jour un état, TD utilise la règle de mise à jour suivante :

$$V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \qquad (30)$$

Alors que la cible du MC était R_t, la cible de TD est $r_{t+1} + \gamma V(s_{t+1})$. Les équations 13 et 15 peuvent aussi s'écrire :

$$V^{\pi}(s) = E_{\pi} \{ r_{t+1} + \gamma V^{\pi}(s_{t+1}) \mid s = s_t \} \qquad (31)$$

La figure 23 donne l'algorithme TD(0)[5].

Initialize $V(s)$ arbitrarily, π to the policy to be evaluated
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    $a \leftarrow$ action given by π for s
    Take action a; observe reward, r, and next state, s'
    $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$
    $s \leftarrow s'$
  until s is terminal

Figure 23: TD(0) pour estimer V^{π}.

[4] En anglais « Temporal Difference » (TD), notation que nous utiliserons dans la suite.
[5] TD est paramétrée par λ. TD(λ) ne sera pas présentée dans ce cours. λ=0 est le cas le plus simple.

La figure 24 donne le diagramme de back-up de TD(0). La mise à jour de l'état ne dépend que de l'état suivant. Contrairement à la mise à jour complète de la PD qui utilise tous les états suivants de l'état courant, la mise à jour de TD est échantillonnée par l'expérience et ne dépend que d'un seul état suivant.

Figure 24: Le diagramme de back-up de TD(0).

TD sur l'exemple du Grid World:

Sur l'exemple du GridWorld, voici un contrôle effectué avec la méthode des différences temporelles:

```
Temporal Difference Control: debut:
  <^>v    <^>v    <^>v    <^>v    <^>v
  <^>v    <^>v    <^>v    <^>v    <^>v
  <^>v    <^>v    <^>v    <^>v    <^>v
  <^>v    <^>v    <^>v    <^>v    <^>v
  <^>v    <^>v    <^>v    <^>v    <^>v
Temporal Difference Control: iteration 1:
  7.08  10.94   6.25   6.75   4.14
  4.01   4.64   3.64   3.35   2.83
  2.46   2.46   2.08   1.94   1.78
  1.47   1.42   1.32   1.23   1.21
  1.14   1.03   0.99   0.96   0.97
    >     <^>v    <       <^>v    <
   ^       ^       ^       ^       ^
   ^       ^       ^       ^       ^
   ^       ^       ^       ^       ^
   ^       ^       ^       ^       ^
Temporal Difference Control: iteration 2:
 21.98  24.42  21.98  18.45  16.60
 19.78  21.98  19.78  16.60  14.94
 17.80  19.78  17.80  14.94  13.45
 16.02  17.80  16.02  13.45  12.10
 14.42  16.02  14.42  12.10  10.89
    >     <^>v    <       <^>v    <
   ^>      ^      <^      <       <^
   ^>      ^      <^      <       <^
   ^>      ^      <^      <       <^
   ^>      ^      <^      <       <^
Temporal Difference Control: iteration 3:
 21.98  24.42  21.98  19.42  17.48
 19.78  21.98  19.78  17.80  15.87
 17.80  19.78  17.80  16.02  14.35
 16.02  17.80  16.02  14.42  12.95
 14.42  16.02  14.42  12.98  11.67
    >     <^>v    <       <^>v    <
   ^>      ^      <^      <       <
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
Temporal Difference Control: iteration 4:
 21.98  24.42  21.98  19.42  17.48
 19.78  21.98  19.78  17.80  16.02
 17.80  19.78  17.80  16.02  14.42
 16.02  17.80  16.02  14.42  12.98
 14.42  16.02  14.42  12.98  11.68
    >     <^>v    <       <^>v    <
   ^>      ^      <^      <       <
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
Temporal Difference Control, fin.
```

Sarsa

Dans cette partie nous étudions une méthode TD permettant d'améliorer une politique et la fonction de valeur associée à cette politique, c'est-à-dire une méthode de contrôle TD. La figure (32) part de l'état s_t et de l'action a_t effectuée dans cet état, elle utilise la récompense r_{t+1}, l'état suivant s_{t+1} et son action associée a_{t+1}. Du fait de l'enchaînement état s, action a, récompense r, état s, action a, cette méthode s'appelle S-a-r-s-a.

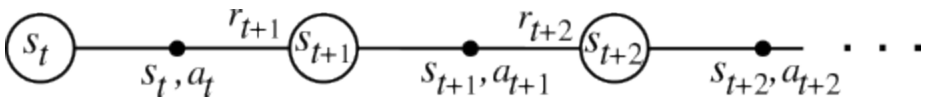

Figure 32

La règle de mise à jour de Sarsa est :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \qquad (32)$$

La figure (33) donne le pseudo-code de l'algorithme Sarsa.

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

Figure 33

Bien remarquer que la politique est absente de cet algorithme car elle est implicitement dictée par Q. Qui plus est, l'action choisie a' correspond à la valeur Q utilisée pour la mise à jour. En ce sens, Sarsa est une méthode vraiement « online ».

Remarquer aussi que le choix de l'action est ε-greedy, sinon la méthode ne marche pas. Si l'on ne souhaite pas avoir de choix ε-greedy, on peut initialiser les valeurs Q avec de *très grandes* valeurs. Ainsi le choix de la meilleure action rendra implicitement la méthode de choix exploratoire.

Q-learning

L'une des avancées les plus importantes en AR est l'algorithme Q-learning de Watkins. La règle de mise à jour de Q-Learning est:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (33)$$

La figure (36) donne le pseudo-code de Q-learning et la figure (37) donne le diagramme de back-up.

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha\left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$
    $s \leftarrow s'$;
  until s is terminal

Figure 36: Q-learning: un algorithme de contrôle TD plutôt "on-policy" et légèrement "off-policy"

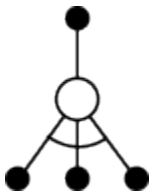

Figure 37: Le diagramme de back-up de Q-Learning.

Bien remarquer que la politique est absente de cet algorithme car elle est implicitement dictée par Q (comme pour Sarsa). En ce sens QL est aussi « on-line ». Cependant, l'action choisie a' n'est pas forcément l'action optimale correspondant à la valeur Q utilisée pour la mise à jour. En ce sens, QL est une méthode légèrement « offline ».

Remarquer aussi que le choix de l'action est encore ε-greedy, sinon la méthode ne marche pas. Si l'on ne souhaite pas avoir de choix ε-greedy, on peut initialiser les valeurs Q avec de *très grandes* valeurs, comme pour Sarsa.

Exemple

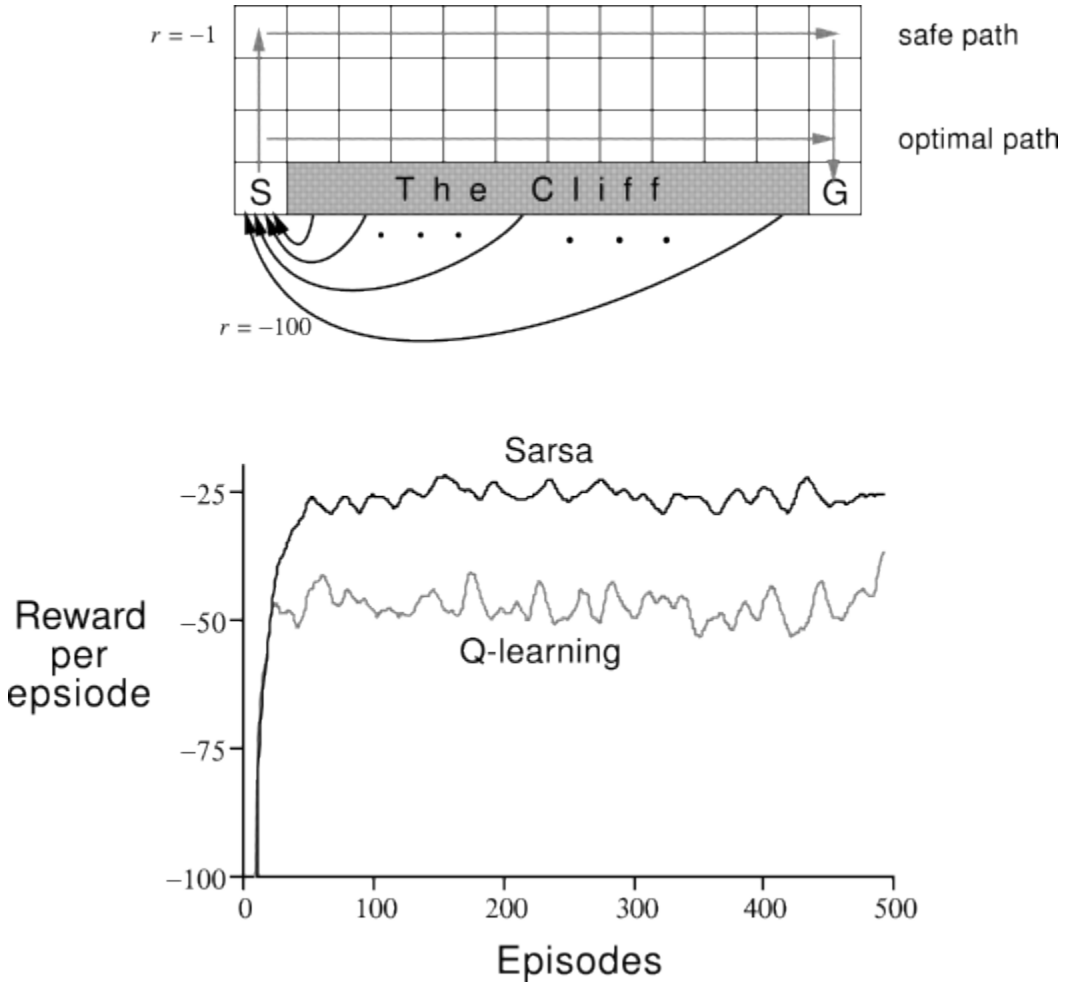

Figure 40 : Le problème de la marche près du ravin. Les résultats proviennent d'une exécution unique mais lissée.

On notera que Q Learning et SARSA se ressemblent beaucoup. Les deux méthodes sont essentiellement « on line »: la politique de choix d'action suit les valeurs de la fonction Q. On peut programmer QL et SARSA sans utiliser de table de politique; Q suffit. Il y a une petite différence entre Q Learning et SARSA. Dans le cas de SARSA, la méthode est exactement on line: la formule de mise a jour suit exactement le choix de l'action (qui n'est pas toujours optimale). Dans le cas de Q Learning, la méthode est presque on-line: la formule de mise à jour utilise la valeur optimale des actions possibles après l'état suivant, alors que comme dans SARSA l'action choisie après l'état suivant peut ne pas être optimale. Cette petite différence entre SARSA et Q Learning rend Q learning un peu plus efficace.

Q Learning sur l'exemple du Grid World:

Sur l'exemple du GridWorld, voici un contrôle effectué avec « Q Learning »:

```
Q Learning: debut:
20 20 22 20 |25 25 25 25 |22 20 20 20 |21 22 22 21 |20 18 18 18 |
18 21 20 18 |20 22 20 20 |20 20 18 18 |18 20 18 18 |18 18 16 16 |
17 19 18 17 |18 20 18 18 |18 18 16 16 |17 18 16 16 |16 17 15 15 |
15 17 16 15 |16 18 16 16 |17 17 15 15 |16 16 14 14 |15 16 14 14 |
14 16 15 14 |14 16 14 14 |15 16 14 14 |14 15 13 13 |14 14 13 12 |
Policy:
    >     <^>v    <         >     <
   ^       ^      <^       ^      <^
   ^       ^       ^       ^       ^
   ^       ^       ^       ^       ^
   ^       ^       ^       ^       ^
delta = 12.544 improve = 96 nb_pas_total = 10000
19 19 22 19 |24 24 24 24 |22 19 19 19 |20 20 20 20 |18 16 16 16 |
17 20 20 17 |19 22 19 19 |20 20 17 17 |18 18 16 16 |16 16 14 14 |
16 18 18 15 |17 20 17 16 |18 18 15 15 |16 16 14 14 |15 15 13 13 |
14 16 16 14 |15 18 15 15 |16 16 14 14 |14 15 13 13 |13 14 12 12 |
13 15 14 13 |14 16 14 14 |14 14 12 12 |13 13 12 11 |12 13 11 11 |
Policy:
    >     <^>v    <       <^>v    <
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^       ^
   ^>      ^      <^       ^       ^
   ^       ^      <^       ^       ^
delta = 2.286 improve = 20 nb_pas_total = 20000
19 19 22 18 |24 24 24 24 |22 19 18 18 |19 19 19 19 |17 15 15 15 |
17 20 20 17 |18 22 18 18 |20 20 17 16 |18 17 15 15 |16 16 14 14 |
15 18 18 15 |16 20 17 16 |18 18 15 15 |16 16 13 13 |14 14 12 12 |
14 16 16 13 |15 18 15 15 |16 16 13 13 |14 14 12 12 |13 13 11 11 |
12 14 14 12 |13 16 13 14 |14 14 12 12 |13 13 11 11 |12 12 10 10 |
Policy:
    >     <^>v    <       <^>v    <
   ^>      ^      <^      <       <
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
delta = 1.058 improve = 16 nb_pas_total = 30000
19 19 22 18 |24 24 24 24 |22 19 18 18 |19 19 19 19 |17 15 15 15 |
17 20 20 16 |18 22 18 18 |20 20 16 16 |18 17 15 15 |16 16 14 13 |
15 18 18 15 |16 20 16 16 |18 18 15 14 |16 16 13 13 |14 14 12 12 |
13 16 16 13 |15 18 15 15 |16 16 13 13 |14 14 12 12 |13 13 11 11 |
12 14 14 12 |13 16 13 13 |14 14 12 12 |13 13 11 11 |12 12 10 10 |
Policy:
    >     <^>v    <       <^>v    <
   ^>      ^      <^      <       <
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
delta = 0.747 improve = 0 nb_pas_total = 40000
Q Learning: fin.
```

Conclusion

Sous la forme d'un tableau, cette conclusion dresse un bilan des méthodes résolvant le « problème de l'AR » défini à la fin de la première partie du cours.

Les méthodes sont:

la résolution directe des équations de Bellmann,
la Programmation Dynamique (PD),
Monte-Carlo (MC),
Rmax,
la méthode des Différences Temporelles (DT)
QLearning (QL) ou Sarsa.

METHODE:	***Résolution équations***	***PD***	***Rmax***	***MC***	***DT***	***QL Sarsa***
itérative ?	directe	itérative	Itérative	itérative	itérative	itérative
visite états	ordonnée	ordonnée	Expérim.	Expérim.	Expérim.	Expérim.
fréq. mise à j.	pas	pas	pas	épisode	pas	pas
Modèle env. ?	Oui	Oui	Construit	Non	Non	Non
retour ?	r	r	r	R	r	r
fonction:	V et/ou Q	V et/ou Q	V ou Q	V et/ou Q	V	Q
type problème	petit	moyen	ass. gros	gros	très gros	très gros
off/on line?		«off»	«on»	«off»	«on»ou«off»	«on»

Figure 41: un tableau récapitulatif des méthodes de résolution du problème de l'AR.

Contrairement à la résolution des équations de Bellmann, la programmation dynamique est une méthode itérative avec les algorithmes « policy evaluation », « policy improvement », « policy iteration » et « value iteration ». Rmax, MC, TD et QL sont aussi des méthodes itératives. MC reprend ces méthodes et correspond à « policy iteration » ou à « value iteration ». TD correspond à « policy evaluation » mais aussi à « value iteration ». QL correspond à « value iteration ».

Contrairement à la programmation dynamique qui balaie les états dans un ordre immuable, et contrairement à la résolution directe des équations de Bellman pour lesquelles une équation correspond à un état, la principale caractéristique des méthodes d'AR est de laisser l'expérience guider la mise à jour des états.

La méthode de MC met à jour les valeurs d'états ou d'actions à la fin d'un épisode avec le retour cumulé. Toutes les autres méthodes « bootstrappent » au sens où elles mettent à jour les valeurs après chaque pas (changement d état) avec le retour instantané.

La résolution directe des équations, la programmation dynamique, Rmax et MC mettent à jour indifférement V ou Q, au choix. En revanche, TD s'applique à V et QL à Q.

Evidemment, l'intérêt relatif de ces méthodes va avec la taille du problème à résoudre: la résolution des équations de Bellman s'applique aux « petits » problèmes pour lesquels une inversion directe de matrice est possible. La programmation dynamique s'applique aux problèmes de taille « moyenne »: tous les états doivent être explicitement en mémoire. Rmax s'applique aux problèmes « assez gros »: plus gros que ceux de la programmation dynamique mais pas autant que TD, QL ou Sarsa qui peuvent s'appliquer au « gros » problèmes. En effet, Rmax construit en mémoire un modèle de l'environnement alors que TD, QL et Sarsa ne le font pas.

Enfin, on peut débattre sur l'aspect « on-line » ou « off-line » des algorithmes d'AR. MC est plutôt off-line mais on peut avoir une version online. TD est l'un ou l'autre. SARSA et QL sont plutôt onlines.

Ce que ce cours n'a pas traité: les « traces d'éligibilités » et TD(λ).

Références

Ronen Brafman, Moshe Tennenholtz, “Rmax – A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning”, Journal of Machine Learning Ressearch, pages 213-231, 2002.

Rémi Coulom, “Apprentissage par renforcement utilisant les réseaux de neurones, avec des applications au contrôle moteur”, Ph.D. thesis, INPG, Grenoble, 2002.

Richard Sutton, Andrew Barto, “Reinforcement Learning”, MIT Press.

Richard Sutton, “Learning to Predict by the methods of Temporal Differences”, Machine Learning 3, pages 9-44, Kluwer, 1988.

Christopher Watkins, “Learning from delayed rewards”, Ph.D. thesis, Cambridge University, 1989.

Christopher Watkins, Peter Dayan, “Q-learning”, Machine Learning, 8, pages 279-292, 1992.