Apprentissage par renforcement

Grid Worlds
DM 1 : MDP

Présentation

Dans ce TP, nous allons travailler sur la mise en pratique des processus de decision markoviens et de la recherche d'une politique optimale tels que vus en cours. Pour cela, nous allons nous intéresser au problème suivant : un agent se trouve dans une grille (GridWorld) $\mathtt{n} \times \mathtt{m}$. Dans la grille, se trouve un puit, donc à éviter, un mur, donc à contourner et une récompense, donc à trouver.

La figure suivante présente une configuration de départ du jeu, avec une grille 4×4.

			E
		B	
H			
	X		

FIGURE 1 – Une GridWorld 4×4 : E est la case à atteindre (end), H est le puit (hole), B est un mur (block) et X est la position actuelle de l'agent.

Les actions possibles pour l'agent correspondent aux déplacements décrits par la figure suivante et correspondent à des pas UP, DOWN, RIGHT ou encore LEFT. Bien sûr, tous ces pas ne sont pas toujours possibles. Par exemple, il suffit que l'agent se trouve sur une des cases de l'extrême gauche de la grille pour qu'un pas LEFT lui soit impossible.

	↑	
←	X	→
	↓	

FIGURE 2 – Pas possibles pour l'agent à partir de sa position

Pour réaliser ce tp, nous disposons d'une classe `Game.py`. Ouvrez ce fichier, parcourez l'ensemble des méthodes de la classe et familiarisez vous avec. En particulier, utilisez quelques une des méthodes pour instancier un jeu (constructeur `Game` avec les bons paramètres), visualiser l'état de la grille (`print()`) et déplacer votre agent (méthode `move()`).

Parcourez également la méthode `move()` pour comprendre les récompenses associées aux déplacements de l'agent.

Remarque. La classe `Game.py` sera utilisée aussi ultérieurement pour d'autres algorithmes de RL (en particulier pour le q-learning). La lecture de la méthode `move()` vous permettra de récupérer les informations sur les récompenses que vous pourrez ensuite implémenter à votre guise.

Travail à faire

1. Créez un fichier `main.py` dans lequel vous commencez par importer le module `Game`.
2. Écrivez une fonction `mdp(game)` permettant de construire le tuple $(\mathtt{S}, \mathtt{A}, \mathtt{p}, \mathtt{r})$ représentant le processus de décision markovien permettant de modéliser le problème qui nous intéresse.

   Indication : vous n'êtes pas obligés de coder "en dur" les matrices des transitions, et des récompenses (comme fait dans l'exemple jouet vu en cours). Vous pouvez également implémenter cela par des méthodes adéquates.
3. Adaptez la fonction `policy_iteration(mdp)` vue en TP pour trouver la meilleure politique pour l'agent.