Apprentissage par Renforcement

M2 DAC

TME 2. Programmation Dynamique

Ce TME a pour objectif d'expérimenter les modèles algorithmes de programmation dynamique sur un MDP classique de type GridWorld.
Nous utiliserons au cours des TME de RL la plateforme en python `gym` (de *open-ai*). Pour l'installer, tapez la commande:

```
pip3 install gym --proxy proxy:3128 --user
```

1 Gym et GridWorld

L'environnement `gridworld-v0` pour `gym` (fourni dans le fichier associé au TME) est une tâche de RL où un agent (point bleu) doit récolter des éléments jaunes dans un labyrinthe 2D et terminer sur une case verte, tout en évitant les cases roses (non terminales) et rouges (terminales). Ci-dessous quelques fonctions utiles pour les environnements :

- initialiser un environnement : `env = gym.make('gridworld-v0')` puis `env.reset()`
- `env.action_space` permet de connaître l'ensemble des actions possibles;
- `obs,reward,done,info = env.step(action)` permet de jouer l'action passée en argument : `obs` contient le nouvel état, `reward` la récompense associée à l'action, `done` un booléen pour savoir si le jeu est fini;
- `env.render()` permet de visualiser le jeu (si env.verbose est à `true`);

Pour le cas du gridworld la fonction `env.getMDP()` retourne un couple `(states,P)`, où:

- `states` est un tableau associant des indices d'états à des états (sous forme de chaînes de caractères);

- `P` correspond à l'automate du MDP de la tâche : chaque clé est un numéro d'état, chaque valeur un dictionnaire; pour ce 2ème dictionnaire, chaque clé est une action et la valeur une liste de tuples correspondant à la transition associée `(proba de la transition,numéro d'état destination,reward,done)`, où `done` est vrai si l'état de destination est un état terminal. Ainsi, `P[state][action]` permet de connaître pour un numéro d'état et une action la liste des états atteignables avec leur probabilité et la récompense associée.

La fonction `env.getStateFromObs(obs)` permet de retourner le numéro d'état d'une observation `obs` (avec `obs` un tableau numpy représentant la carte correspondant à l'observation de l'état). Cette fonction est utile pour convertir l'observation retournée par `env.step(action)` en numéro d'état plus facile à manipuler.

Le répertoire `gridworld` contient l'environnement du labyrinthe; le fichier `randomAgent.py` présente des exemples d'utilisation de l'environnement et de `gym` et en particulier un agent aléatoire.

Dans l'environnement `gridworld` il est possible de charger différentes cartes de problème par la fonction `setPlan("gridworldPlan/planX.txt",{0:-0.001,3:1,4:1,5:-1,6:-1})` prenant en argument le fichier de la carte à charger et une liste des récompenses associées aux différents types de cases du jeu : 0 correspond à une case vide, 1 correspond à un mur (pas de reward associé car impossible de s'y déplacer), 2 correspond au joueur, 3 correspond à une case verte, 4 une case jaune, 5 une case rouge et 6 une case rose. Dans le module fourni, vous trouverez différentes cartes de jeu (`X` de 0 à 10), de difficulté variable, que vous devrez tester au cours du TME.

2 Travail demandé

Codez les algorithmes de policy iteration et de value iteration vus en cours et testez les sur l'environnement. Discutez des résultats obtenus sur les différentes cartes/configurations (vous pouvez aussi en imaginer d'autres). Vous pouvez aussi faire varier le paramètre de discount pour en observer l'impact.