Auteurs : Juliana Sanchez et Maxime Herry **4TC1** INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Github Herry Sanchez

CR - IAT

1. Résumé du travail.

(a) Caractéristiques des algos et code

Nous avons suivi le TD et avons pu implémenter l'algorithme de programmation dynamique dans un premier temps, puis le Q-learning dans un second temps.

Vous pourrez trouver tous les codes sur github (en haut de chaque page) : https://github.com/Larobase/IAT-Herry-Sanchez

i) Programmation dynamique

La théorie de la programmation dynamique a été découverte en 1952 par Richard Bellman. L'algorithme de programmation dynamique est une méthode basée modèle, qui repose sur une application itérative de l'opérateur de Bellman et qui permet l'amélioration d'une politique, donc retourne la meilleure action à faire dans un état donné. Ceci revient à décomposer un problème en sous problèmes pour les résoudre du plus petit au plus grand.

Dans notre cas, nous initialisons un dictionnaire avec des valeurs nulles pour chaque état. Ensuite, tant que la différence entre ce dictionnaire sur deux itérations est plus grande qu'un seuil, on continue de le mettre à jour. Une fois qu'il a fini, on choisit la politique de chaque état. Pour cela, on prend un état, on regarde ses voisins, et on lui dit d'aller sur le voisin avec la plus grande valeur, pondéré par le fait de se rapprocher d'une bonne ou mauvaise récompense.

ii) Q-learning

La méthode du Q-learning est une méthode d'apprentissage par renforcement. C'est une méthode indirecte sans modèle qui a plusieurs propriétés :

1. La convergence vers Q* avec une représentation tabulaire
2. La divergence avec une représentation basée Q-Networks
3. La corrélation entre les échantillons
4. La non stationnarité des cibles

Dans notre cas, on initialise un dictionnaire avec des valeurs nulles pour chaque état et pour chaque direction par état. Ensuite, on effectue un nombre d'épisodes prédéfinis. Pour chaque épisode, on part de l'état initial et on ne s'arrête pas tant que l'on n'est pas arrivé sur un état terminal. On mémorise le chemin parcouru en modifiant les valeurs de Q [s][a], pondéré par le learning rate. Ensuite, pour choisir la politique, on prend la direction avec la valeur la plus élevée pour chaque état.

(b) Convergence des algorithmes ?

i) Programmation dynamique

La condition de cet algorithme pour s'arrêter est que la différence entre son calcule de V actuel et son précédent soit inférieur à epsilon, un paramètre modifiable. Pour calculer chaque V, l'algorithme ne parcourt pas de chemin mais calcule la valeurs des états en fonction du modèle qui existe déjà donc même si le chemin est bloqué vers les états terminaux, nous pourrons observer des états avec des valeurs différentes de 0 autour des états terminaux.

ii) Q-learning

La condition de cet algorithme pour finir un épisode est de trouver un état terminal, ainsi, si on bloque le passage jusqu'aux états terminaux, l'algorithme ne converge jamais et ne peut même pas finir un épisode.

En théorie le Q-learning converge, mais ce n'est pas forcément le cas en pratique car les poids mettent à jour alpha, donc ce n'est pas stationnaire. De plus, si les réseaux sont corrélés, la loi des grands nombre n'est plus valide.

2. Résultats d'analyse

(a) Preuves expérimentales de convergence, divergence et autres comportements

i) Preuve de convergence

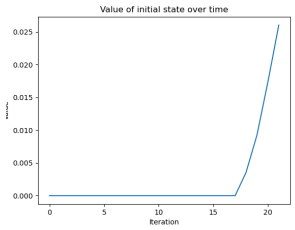
Pour commencer, nous pouvons montrer qu'il existe bien un affichage pour la programmation dynamique, quand les états terminaux sont inaccessibles depuis le départ, ce qui, nous le rappelons est impossible avec le Q-learning. Ici, tous les autres états sont bien à 0 car il n'y a pas de chemin vers une quelconque récompense.

0.00	0.00	0.00	0.00	0.00	0.00	0.00	#	0.85	1.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	#	0.57	-1.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	#	0.48	0.28
0.00	0.00	0.00	0.00	0.00	0.00	0.00	#	#	#
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

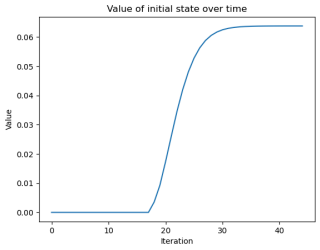
Figure 1 : Preuve de convergence programmation dynamique

De plus, nous avons pu montrer la convergence en mesurant la valeur des états initiaux en fonction du temps comme dans les figures ci-dessous.

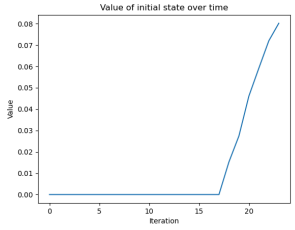
Github Herry Sanchez



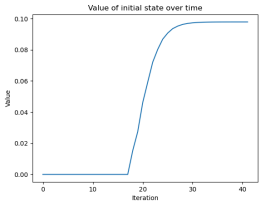
a) epsilon = 0.01, 50 murs



b) epsilon = 0.000001, 50

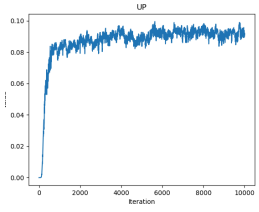


c) epsilon = 0.01, 1 mur

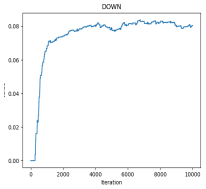


d) epsilon = 0.000001, 1 mur

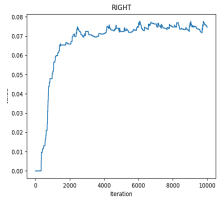
Figure 2 : Convergence de V[(0,0)], grid 10*10
Programmation dynamique



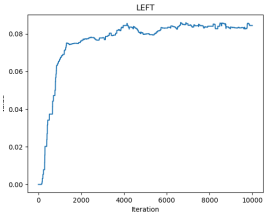
a) UP



b) DOWN



c) RIGHT



d) LEFT

Figure 3 : Convergence de Q[(0,0)][a] pour a étant chaque direction, grid 10*10, 1 mur, 10 000 époques
Q-learning

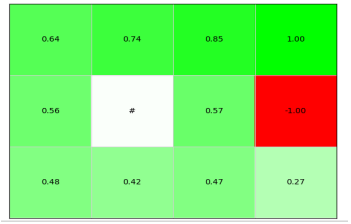
Github Herry Sanchez

ii) Visualisations graphiques

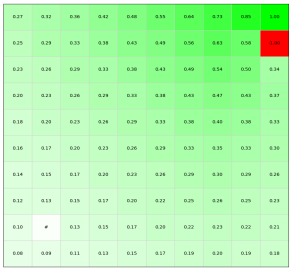
Ensuite, nous pouvons nous focaliser sur la programmation dynamique. Nous avons pu faire tourner l'algorithme en changeant plusieurs paramètres :

- la taille de la grille
- le nombre de murs sur la grille (nous avons demandé à une IA de nous générer 50 tuples aléatoires)
- l'hyperparamètre epsilon

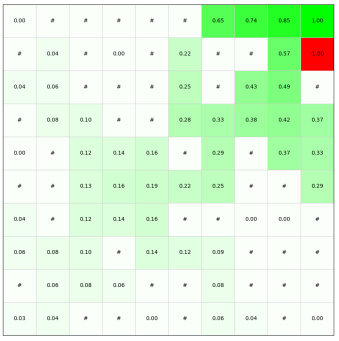
Dans les Figure 4, nous pouvons observer la valeur attribuée à chaque état puis sur la figure 5 la politique choisie sur chaque état. Par défaut, quand les voisins ont les mêmes valeurs, la direction choisie est vers le haut. Nous pouvons remarquer que l'algorithme arrive à se frayer un chemin à travers les murs en trouvant celui qui mène à la case finale. En augmentant epsilon, la propagation de la récompense sur la grille est moins grande. Nous constatons que les valeurs sont plus élevées près de l'état terminal, de récompense +1. Néanmoins, les états voisins de l'état terminal de récompense -1 sont affectés et ont une valeur plus faible que les autres états équidistants de l'état +1.



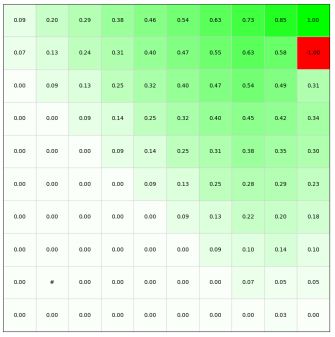
a) grid 4*3, 1 mur, epsilon = 0.01



b) grid 10*10, 1 mur, epsilon = 0.01



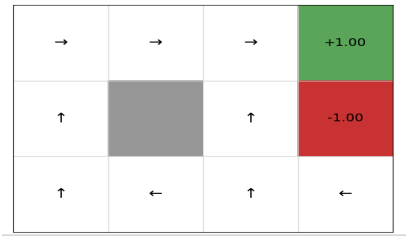
c) grid 10*10, 50 murs, epsilon = 0.01



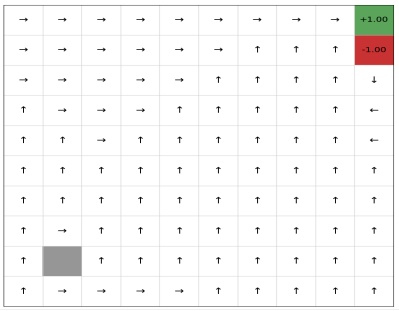
d) grid 10*10, 1 mur, epsilon = 0.1

Figure 4 : Value function en heatmap
programmation dynamique

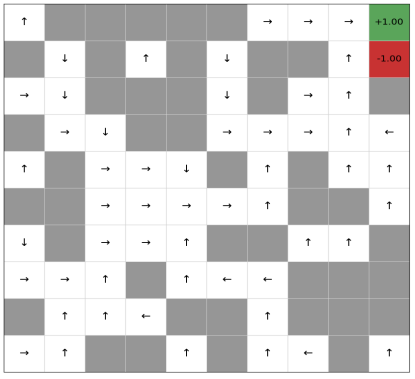
Github Herry Sanchez



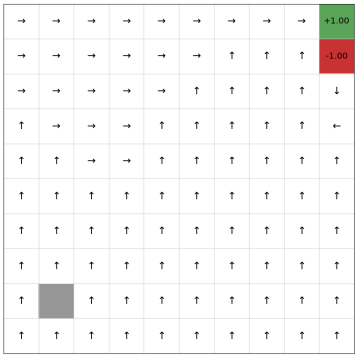
a) grid 4*3, 1 mur, epsilon = 0.01



b) grid 10*10, 1 mur, epsilon = 0.01



c) grid 10*10, 50 murs, epsilon = 0.01



d) grid 10*10, 1 mur, epsilon = 0.1

Figure 5 : Politique

Programmation dynamique

Ensuite, nous pouvons nous attarder sur le Q-learning. Dans notre cas, nous avons pu faire varier les mêmes paramètres que la programmation dynamique, en ajoutant le paramètre alpha. Nous remarquons que le Q-learning a beaucoup plus de mal à converger quand il y a plus de 10 murs.

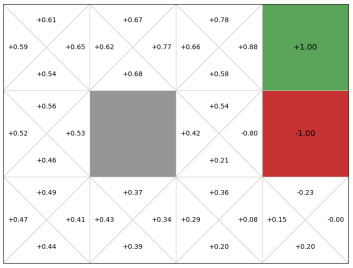
En faisant augmenter epsilon, on augmente le pourcentage de chance de choisir une direction aléatoire pour chaque déplacement donc il est plus probable d'aller sur des états encore jamais visités et l'exploration est plus vaste. On voit bien que quand epsilon = 1, la répartition du nombre de passage par état est uniforme. Si epsilon est trop faible, l'algorithme se bloque dans un minimum local et doit tomber sur une chance infime de faire aléatoirement un autre déplacement découvrir la vraie solution comme sur la figure 7.2 a).

En augmentant alpha, le learning rate, on accorde plus de poids aux estimations de la fonction Q. Pour un alpha trop petit, il y a le risque de tomber dans un optimum local, ce qui explique la Figure 7.2 d). Le parcours actuel a moins d'impact sur la mise à jour des estimations. A l'inverse, un alpha trop grand crée des instabilités qui peuvent aller jusqu'à une divergence.

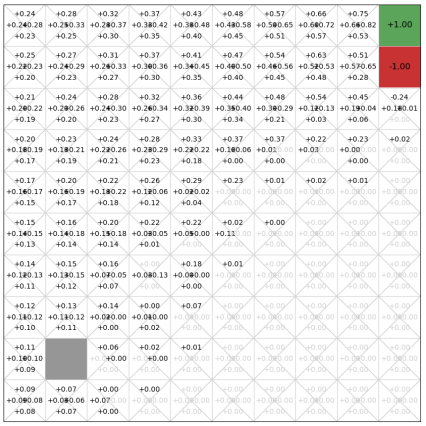
Auteurs : Juliana Sanchez et Maxime Herry **4TC1**

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

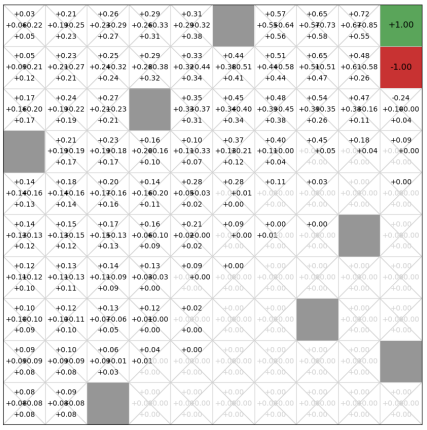
Github Herry Sanchez



a) grid 4*3, 1 mur

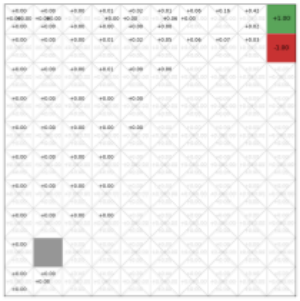


b) grid 10*10, 1 mur

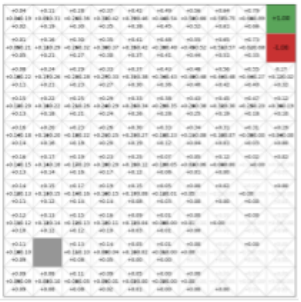


c) grid 10*10, 7 murs

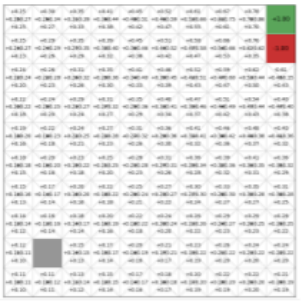
Figure 6.1 : Q Values, 10 000 époques, epsilon = 0.1, alpha = 0.1
Q-learning



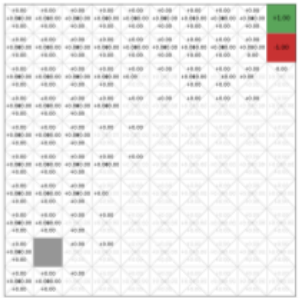
a) alpha = 0.1, epsilon = 0.0001



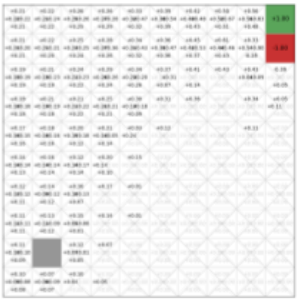
b) alpha = 0.1, epsilon = 0.5



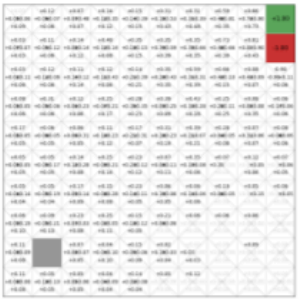
c) alpha = 0.1, epsilon = 1



d) alpha = 0.0001, epsilon = 0.1



e) alpha = 0.5, epsilon = 0.1



f) alpha = 1, epsilon = 0.1

Figure 6.2 : Q Values, 1000 époques, 1 mur, grid 10*10
Q-learning

Github Herry Sanchez

a) grid 4*3, 1 mur

b) grid 10*10, 1 mur

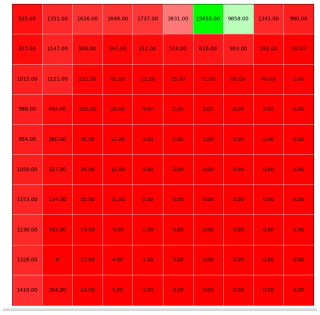
c) grid 10*10, 7 murs

Figure 7.1 : Nombre de passages par état, epsilon = 0.1, alpha = 0.1
10 000 époques
Q-learning

a) alpha = 0.1, epsilon = 0.0001

b) alpha = 0.1, epsilon = 0.5

c) alpha = 0.1, epsilon = 1



d) alpha = 0.0001, epsilon = 0.1

e) alpha = 0.5, epsilon = 0.1

f) alpha = 1, epsilon = 0.1

Figure 7.2 : Nombre de passages par état, grid 10*10, 1000 époques
Q-learning

iii) Visualisations temporelles

Enfin, nous avons pu faire des tests en fonction du temps pour voir à quelle vitesse les algorithmes convergent. Pour la programmation dynamique, plus il y a de murs, plus on converge rapidement car pour les murs le calcul de cette valeur est faite une seule fois (à cause du do while). À l'inverse, pour le Q-learning, plus il y a de murs, plus le temps de convergence est élevé et à partir de 10 murs, cela devient exponentiel.

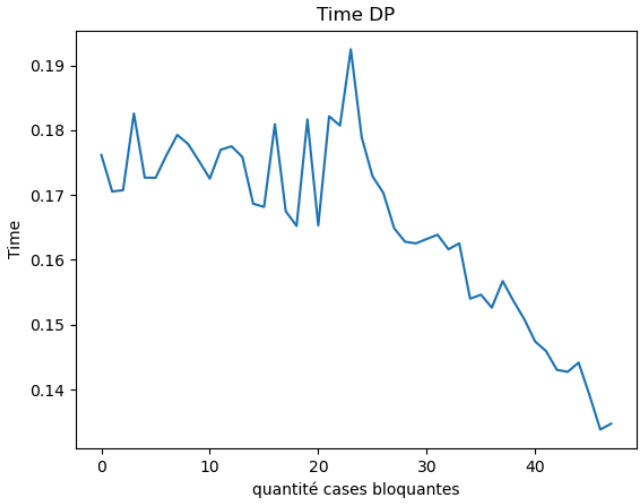


Figure 8 : Temps de convergence en fonction du nombre de murs
Dynamic programming

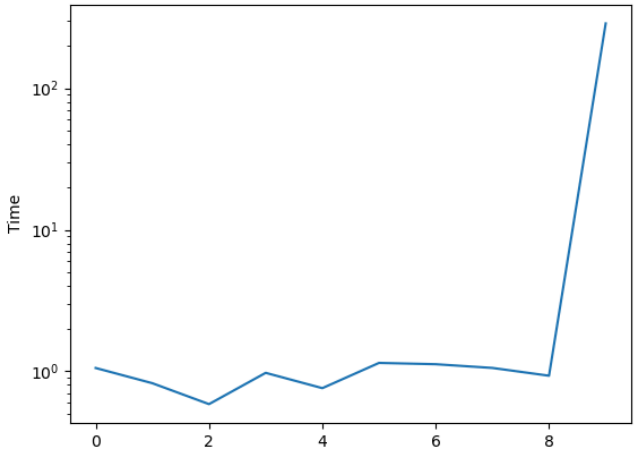


Figure 9 : Temps de convergence en fonction du nombre de murs (de 1 à 10)
en échelle log, 1000 époques
Q-learning

Github Herry Sanchez

Pour essayer de comprendre ce phénomène, nous avons tracé le temps de chaque époque et on peut remarquer que les premières époques prennent ici beaucoup plus de temps que celles d'après. Une fois que l'algorithme à trouvé une piste, il s'y engouffre et continue dans la même voie. en explorant un peu au hasard à côté de temps en temps.

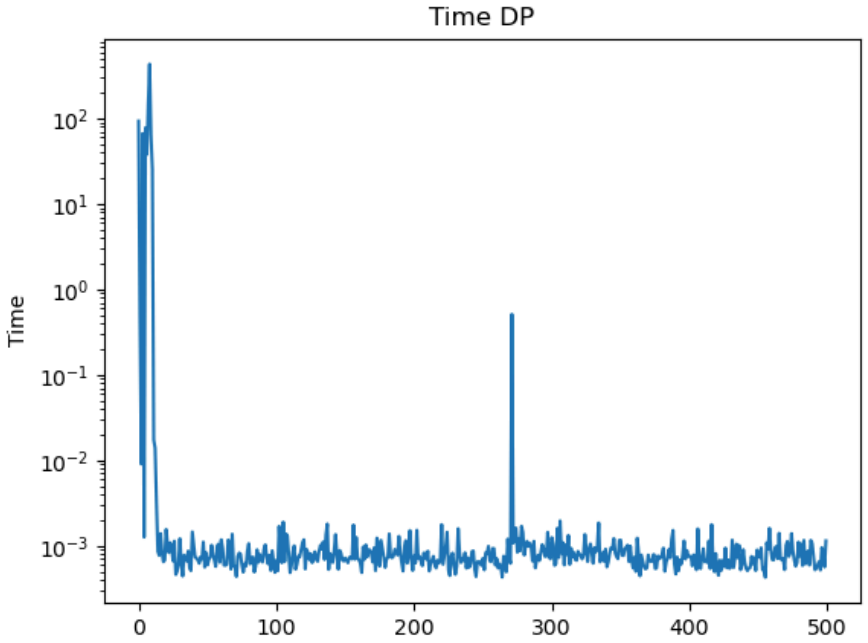


Figure 10 : Temps de convergence des époques pour 10 murs
500 époques
Q-learning

(c) Conclusion sur la différence des deux algorithmes

En conclusion, la programmation dynamique s'est révélée être une méthode plus robuste pour résoudre des problèmes. Nous avons observé sa capacité à converger même lorsque certains états terminaux sont inaccessibles, ainsi que sa performance stable et prévisible dans des environnements avec des structures bien définies.

D'autre part, le Q-learning offre une approche plus flexible pour apprendre à agir dans des environnements inconnus ou complexes. Cependant, nous avons constaté que sa convergence peut être plus longue, en particulier en présence d'obstacles. De plus, des paramètres tels que le learning rate et l'epsilon peuvent avoir un impact important sur sa capacité à explorer l'espace des états.

Finalement, le choix entre la programmation dynamique et le Q-learning dépendra de la nature du problème à résoudre.