TME2 RLD

Programmation Dynamique

Brient, Hein - Décembre 2021

1 Objectifs

Ce TME a pour objectif d'expérimenter certains modèles de programmation dynamique sur un MDP classique de type GridWorld. Dans ce projet, l'environnement de travail est une tâche d'apprentissage par renforcement 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). L'agent doit suivre la politique lui permettant d'obtenir la plus grande récompense, qui se traduit notamment par réussir à ramasser tous les éléments jaunes. Lorsque, comme dans notre cas, le MDP est entièrement connu, cette politique peut être déterminée séparemment et indépendamment des expériences de l'agent. Pour ce faire, nous allons implémenter sur cet environnement deux algorithmes classiques de recherche de politique optimale : Policy Iteration et Value Iteration. Plusieurs cartes sont à disposition pour l'étude, de difficultés et formes variables, permettant de bien appréhender le comportement et les limites de ces algorithmes.

2 Algorithmes

Résoudre ces problèmes correspond à trouver pour chaque état l'action permettant à l'agent d'engrenger en moyenne le plus de récompenses jusqu'à la fin de l'épisode. Ainsi, une notion centrale en programmation dynamique est introduite, la state-value définie pour chaque état s par :

$$V_\pi(s) = \mathbb{E}_\pi(G_t|S_t = s)$$

où on a défini :

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$$

$V_\pi(s)$ est définie pour chaque état s et pour chaque itération t et est toujours basée sur une politique à suivre π. G_t correspond à une somme pondérée des récompenses instantanées R obtenues à chaque itération jusqu'à la fin de l'épisode. Le paramètre γ permet de demander à notre modèle de mettre plus ou moins de poids sur les récompenses futures, ce qui dépend de l'expérience.

Par ailleurs, les travaux de Bellman quantifient le lien entre les state-value des états aux itérations t et $t+1$ par l'équation suivante :

$$V_\pi(s) = \mathbb{E}_\pi(R_{t+1+k} + \gamma V_\pi(s')|S_t = s)$$

Ce qui donne après développement :

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r P(s', r|s, a)[r + \gamma V_\pi(s')]$$

2.1 Policy Evaluation and Improvement

Un point central dans la recherche de politique optimale est l'évaluation d'une politique donnée. L'équation précédente est légèrement modifiée pour obtenir :

$$V_\pi(s) = \sum_a \pi(a|s)[R(s, a) + \gamma \sum_{s'} P(s'|s, a)V_\pi(s')]$$

Pour éviter d'avoir affaire à des temps de calculs trop longs, cette affectation est remplacée par un processus itératif issu de la programmation dynamique :

$$V_{k+1}(s) = \sum_a \pi(a|s)[R(s, a) + \gamma \sum_{s'} P(s'|s, a)V_k(s')]$$

Puisque V_k tend vers V_π lorsque k tend vers l'infini, on peut alors simplement implémenter un processus itératif permettant d'évaluer les state-value induits par une politique donnée. Tout cela est utile car deux politiques peuvent être comparées via leurs state-values. En effet la politique π' est meilleure que la politique π à l'état s si on a :

$$V_{\pi'}(s) > V_\pi(s)$$

Une des méthodes d'amélioration de la politique est alors d'agir de façon gloutonne en changeant toutes les actions dans tous les états en choisissant pour chaque état s, l'action a telle que :

$$\forall s, \pi'(s) = argmax_a(Q_\pi(s, a))$$

où on a défini :

$$Q_\pi(s, a) = \mathbb{E}_\pi(G_t|S_t = s, A_t = a)$$

Cette méthode choisit l'action qui semble la meilleure à seulement une profondeur égale à 1, mais à chaque itération la méthode assure l'obtention d'une meilleure politique, jusqu'à convergence vers la politique optimale.

2.2 Policy Iteration

L'algorithme de policy itération découle directement de ces calculs. Jusqu'à convergence vers la politique optimale, l'algorithme va à tour de rôle évaluer la politique actuelle puis l'améliorer selon le principe précédent, à l'aide des state-value calculés dans la phase d'évaluation. Ainsi en développant l'expression précédente on modifie la politique pour l'ensemble des états suivant l'affectation suivante :

$$\forall s, \pi'(s) = argmax_a(\sum_a P(s'|a, s)[R(s', a, s) + \gamma V_\pi(s')])$$

2.3 Value Iteration

Pour améliorer le processus de convergence de l'algorithme de Policy Iteration vers la politique optimale, l'algorithme de Value Iteration utilise les relations suivantes :

$$\forall s, V^*(s) = V_{\pi^*}(s) = max_\pi V^\pi(s) = max_a(\sum_{s'} P(s'|a,s)[R(s',a,s) + \gamma V^*_\pi(s')])$$

Ainsi l'algorithme de Value Iteration va chercher à trouver pour chaque état la state value optimale pour ensuite déterminer la politique optimale sans modifier plusieurs fois la politique. La première étape va consister à faire converger itérativement les state-value vers leurs valeurs optimales selon l'affectation suivante :

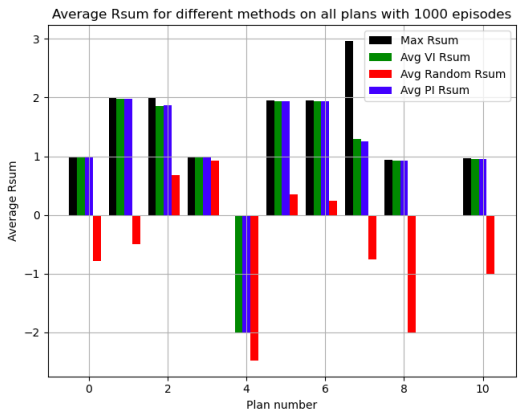
$$V_{k+1}(s) = max_a(\sum_{a} P(s'|a,s)[R(s',a,s) + \gamma V_k(s')])$$

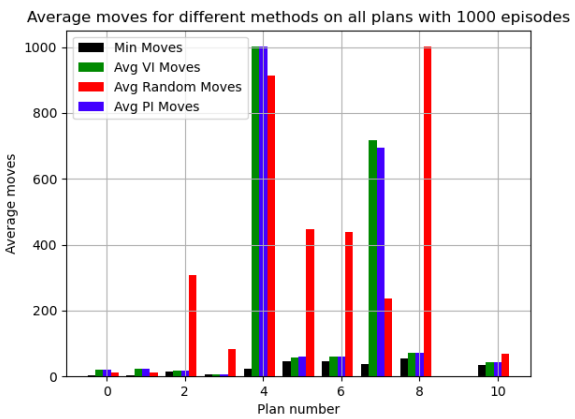
La seconde met à jour la politique une seule fois en fin d'algorithme grâce à la même relation que pour l'algorithme de Policy Iteration.

3 Implémentation et résultats

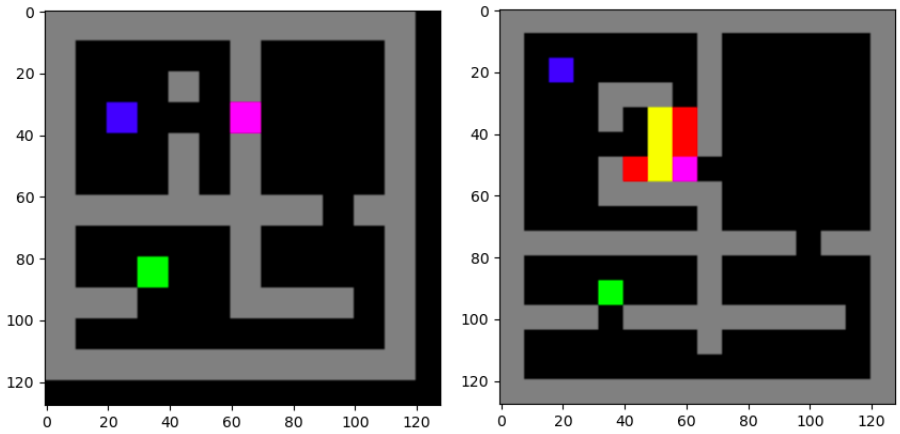
Nous avons implémenté puis testé ces deux algorithmes et les avons comparé avec un agent Random sur tous les plans utilisables. Le paramètre ϵ, a été fixé pour l'ensemble des expériences à 10^{-7} pour assurer une bonne convergence lors des processus itératifs respectifs des deux algorithmes. Cette valeur est faible pour éviter que deux politiques très semblables lors du processus de convergence puissent arrêter l'itération. Par ailleurs, plusieurs expériences ont été faites pour mesurer l'influence du paramètre de discount γ.

Sur le graphique de gauche sont recensées les récompenses moyennes de chaque agent sur 1000 épisodes, pour chacun des plans. De plus, elles peuvent être comparées à la politique optimale correspondante, représentée en noir par les récompenses moyennes maximales atteignables sur chaque plan. Sur le graphique de droite, selon le même principe, sont représentés les nombres moyens de mouvements de l'agent. Les agents Random, de Policy Iteration et de Value Iteration sont respectivement représentés par les couleurs rouge, bleu et verte. Pour la première expérience, la valeur de γ a été fixée à 0.99 et les résultats sont affichés ci-dessous.



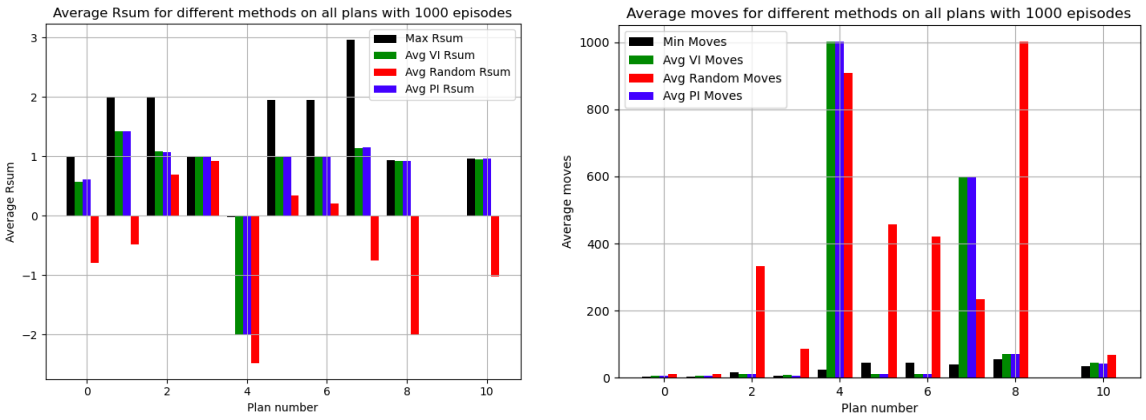


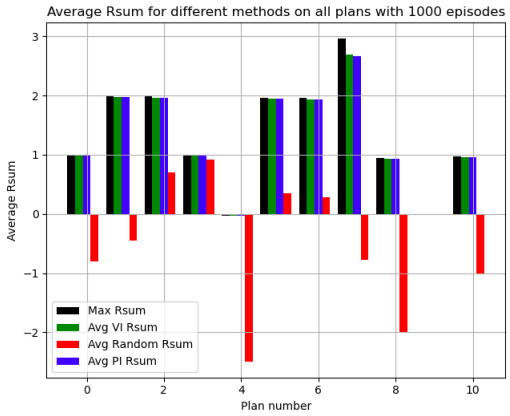
Les deux algorithmes parviennent à trouver rapidement une politique quasi-optimale pour la majeure partie des plans, mais deux plans semblent poser problème : les plans 4 et 7. Pour le plan 4, les agents ne parviennent pas à trouver leur chemin, en témoigne les itérations moyennes atteignant le seuil maximum autorisé, signe qu'aucune politique satisfaisante n'a été trouvée. Par contre, pour le plan 7, les politiques semble ne pas saisir certaines subtilités de la carte puisque toutes les récompenses ne sont pas récupérées et les agents arrivent à destination avec un nombre d'itérations très loin de celui optimal. En les affichant ci-dessous, on peut se rendre compte des problèmes causés par ces deux plans.

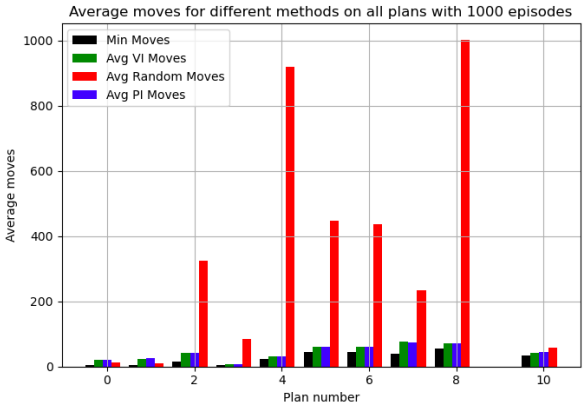


Le plan 4 à gauche oblige l'agent à passer par une récompense négative, le forcant, si les rewards futures ne sont pas assez elevées, à rester près de la zone de départ jusqu'à atteindre le taux maximal d'itérations. Le même soucis est présent dans le plan 7 à droite, mais à moindre mesure. En effet, si l'on ne met pas assez de poids sur les récompenses futures, jamais la case verte, située relativement loin de l'agent, ne va être assez rentable pour compenser l'obligation de passer par une case rose dans les itérations proches, d'où l'intérêt de bien configurer le paramètre γ.

Pour s'assurer de ces propos et tenter de palier à ces difficultés, les mêmes expériences ont été réalisées respectivement avec un discount de 0.9 et 0.999 et sont affichées ci-dessous.







A l'aide de ces graphiques supplémentaires, on peut saisir l'importance du paramètre γ. On peut notamment remarquer qu'avec un discount plus faible égal à 0.9, la majorité des plans voient leurs performances baisser. Toutefois, les plans 3, 8 et 10 ne semblent pas être affectés. Par ailleurs, avec un discount plus élevé de 0.999, les deux algorithmes ont pu trouver des politiques performantes et presque optimales pour tous les plans. En conclusion, augmenter le paramètre γ permet d'améliorer les performances sur des plans plus "difficiles" comme les plans 4 et 7, demandant plus de poids sur les récompenses futures. En contre-partie, cela demande évidemment un temps de calcul bien plus élevé qui est alors contre-productif pour des plans plus "simples", comme les plans 3, 8 et 10. Il y a donc un compromis à trouver entre performance et temps de calcul, propre à chaque problème, et ici en l'occurence à chaque plan.

A titre indicatif, le graphique suivant présente la même expérience avec un paramètre γ de 0.9999 et a été obtenu après un long temps de cacul. On peut voir que sur certains plans, les très légères marges de progression ont été comblée mais qu'il n'y a pas eu, ou très peu, d'amélioration de peformances pour le plan 7, ce qui semble illustrer les limites des deux algorithmes. Cependant, il faut aussi prendre en compte que la politique de l'agent est stochastique, ce qui influe évidemment sur les performances en augmentant le nombre d'itérations et donc en diminuant les récompenses totales. Toutefois, les itérations supplémentaires n'expliquent qu'en partie l'écart de performances sur le plan 7 entre une politique optimale et celles trouvées par les deux algorithmes étudiés.