Comparaison de solveurs pour les Processus de Décisions Markovien

Calic Petar, Muyang Shi
Sorbonne Université, Paris
Encadré par Emmanuel Hyon

petarcalic99@gmail.com
muyangforwork@gmail.com

18/04/21

Abstract

Nous avons pour objectif dans ce projet, de faire une comparaison numérique de solveurs pour MDP. Afin de faire cela, nous allons d'abord faire des recherches sur les fondements des MDP pour les maîtriser. Ceci est essentiel puisque nous devrons ensuite prendre en main chacun des 3 solveurs que nous allons étudier. Nous allons ensuite préparer un benchmark qui doit être appliqué à tous les solveurs. La dernière phase est celle ou on doit faire tourner les solveurs sur nos exemples et analyser les résultats obtenus.

Keywords: MDP; solveurs; benchmark

1 Introduction

En théorie des probabilités, un processus de décision markovien (MDP) est un modèle stochastique (de fonction aléatoire) où un agent prend des décisions et où les résultats de ses actions sont aléatoires. Les MDPs sont une extension des chaînes de Markov avec plusieurs actions à choisir par état et où des récompenses sont gagnées par l'agent. Les MDPs sont utilisés pour étudier des problèmes d'optimisation à l'aide d'algorithmes de programmation dynamique ou d'apprentissage par renforcement, que nous allons étudier et comparer.

1.1 Objectifs du Projet:

1.1.1 Maitriser les MDP

Avant de pouvoir manipuler les solveurs, il est crucial de comprendre les fondements théoriques des MDP. Tout d'abord il faut savoir reconnaître les situations dans lesquelles un problème peut être modélisé par un MDP. Des hypothèses telles que l'hypothèse de Markov doivent être vérifiées. Ensuite il va falloir savoir modéliser un problème sous forme de MDP, puis adapter le modèle au solveur qu'on souhaite utiliser.

1.1.2 Prendre en main les Solveurs

L'objectif principal de ce projet est de comparer 3 solveurs de MDP. Nous allons observer les performances de Gurobi(programme mathématique) MarmotMDP et ToolboxMDP. Avant de faire cela il va falloir bien s'approprier les 3 solveurs et savoir les manipuler. Trois grandes familles de méthodes existent pour résoudre de tels MDP en cherchant la politique optimale:

- Value Iteration: L'approche la plus classique qui se base aussi sur la résolution directe de l'équation d'optimalité de Bellman V. On fait des Itérations sur les états jusqu'à convergence.
- Policy Iteration: Itération sur la politique; On propose une politique de départ et chaque itération fait une évaluation puis amélioration de la politique jusqu'à convergence.
- Programmation linéaire: Transformation de l'MDP en un problème linéaire et son Dual pour le résoudre.

Selon les critères, les algorithmes ne sont pas les mêmes à utiliser. Les types de résultats(politiques) ne sont pas les mêmes aussi.

1.1.3 Modéliser un problème MDP pour solveur

Pour pouvoir comparer les solveurs, nous allons prendre un exemple de problème modélisable comme un MDP et le résoudre. Il y a une étape intermédiaire importante et c'est la modélisation du problème et donc son adaptation ou traduction pour un solveur spécifique. En effet, chaque solveur requiert une modélisation particulière d'un problème pour pouvoir le résoudre.

1.1.4 Analyse des solveurs

C'est l'étape finale et clé du projet. Nous allons faire tourner les 3 solveurs sur un même problème et observer les résultats. Nous allons construire un benchmark (Un ensemble d'exemples pour tester les caractéristiques) afin de comparer leurs caractéristiques comme la vitesse d'exécution, l'optimalité de la solution, la précision, la complexité d'utilisation.
Nous allons ensuite faire une synthèse des caractéristiques de chaque solveur et noter leurs avantages et inconvénients. Nous allons déterminer aussi pour chaque type de problème, quel solveur serait le plus adapté et performant.

1.1.5 Planning estimé:

- Recherche et maîtrise des MDP: 01 février - 30 février
- Prise en main des solveurs: 01 mars - 21 mars
- Préparation du benchmark: 21 mars - 14 avril
- Analyse des résultats: 14 avril - 25 avril
- Rédaction du rapport : 25 avril - 10 mai

Cela correspond à 3 mois et demi de travail. Ceci est notre estimation de temps nécessaire pour faire un travail de qualité sans grande pression du temps.
Sous demande du professeur encadrant, l'ensemble du projet sera codé sous Python à l'aide d'une machine virtuelle sous Linux. Le rapport aussi devra être écrit à l'aide de Overleaf en Latex.

2 Présentation des MDP

2.1 Définition formelle

Un MDP est un quadruplet S, A, T, R définissant :

- un ensemble d'états S, qui peut être fini, dénombrable ou continu; cet ensemble définit l'environnement tel que perçu par l'agent (dans le cas d'un robot, on peut voir cela comme l'ensemble produit des valeurs de ses différents capteurs);
- un ensemble d'actions A, qui peut être fini, dénombrable ou continu et dans lequel l'agent choisit les interactions qu'il effectue avec l'environnement (dans le cas d'un robot on peut voir cela comme l'ensemble produit des paramètres de ses différentes commandes);
- une fonction de transition $T : S \times A \times S \to [0; 1]$; cette fonction définit l'effet des actions de l'agent sur l'environnement: $T(s, a, s')$ représente la probabilité de se retrouver dans l'état s' en effectuant l'action a, sachant que l'on était à l'instant d'avant dans l'état s;
- une fonction de récompense $\Re : S \times A \times S \times \Re \to [0; 1]$; elle définit la récompense (positive ou négative) reçue par l'agent:$R(s, a, s')$ est la probabilité d'obtenir une récompense r pour être passé de l'état s à s' en ayant effectué l'action a. Ici encore cette définition est très générale, bien souvent on se contentera par exemple des cas particuliers suivants :
  - $\Re : S \times A \times S \to \Re$ (récompense déterministe, c'est le choix que nous adopterons dans la suite) ;
  - $\Re$: S x A $\to \Re$ (récompense déterministe rattachée à l'action en ignorant son résultat) ;
  - $\Re$: S $\to \Re$ (récompense déterministe rattachée à un état donné).

2.1.1 Illustration d'un MDP:

Soit l'état initial d'un agent s0, on choisit une action a0 dans A à exécuter. Après la exécution, l'agent passe aléatoirement à l'état prochain s1 en fonction de la probabilité de T. Et puis l'action a1 permet de passer à s2...
On peut représenter cette processus comme la figure ci-dessous :

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \dots$$

Figure 1: Exemple d'un MDP simple.

Si la récompense a obtenu par rapport aux états s et aux actions a effectué, le MDP peut être représenté ainsi :

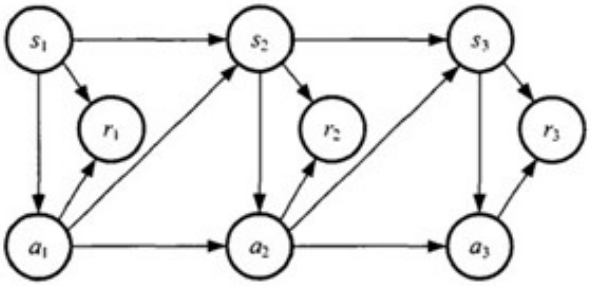

Figure 2: MDP et récompenses.

2.2 Un exmeple de MDP

L'exemple donné ci-dessous représente un processus de Décision Markovien à trois états distincts s_0, s_1, s_2 représentés en vert. Depuis chacun des états, on peut effectuer une action de l'ensemble a_0, a_1. Les nœuds rouges représentent une décision possible (le choix d'une action dans un état donné). Les nombres indiqués sur les flèches sont les probabilités d'effectuer la transition à partir du nœud de décision. Les récompenses sont représentées en jaune.

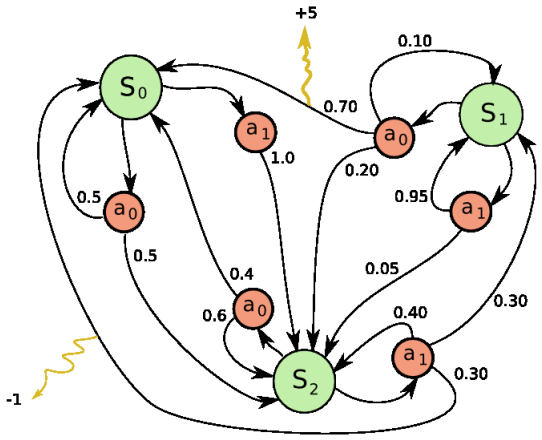

Figure 3: Exemple d'un MDP.

La matrice de transition associée à l'action a_0 est la suivante : $\begin{pmatrix} 0.50 & 0 & 0.50 \\ 0.70 & 0.10 & 0.20 \\ 0.40 & 0 & 0.60 \end{pmatrix}$

La matrice de transition associée à l'action a_1 est la suivante : $\begin{pmatrix} 0.50 & 0 & 0.50 \\ 0.70 & 0.10 & 0.20 \\ 0.40 & 0 & 0.60 \end{pmatrix}$

Les récompenses sont $+5$ lorsque l'on passe de l'état s_1 à l'état s_0 en accomplissant l'action a_0. On perçoit une récompense de -1 (pénalité) lorsque l'on passe de l'état s_2 à l'état s_0 en faisant l'action a_1.

2.3 Les politiques d'actions

Le domaine T des étapes de décision est un ensemble discret, qui peut être fini ou infini (on parle d'horizon fini ou d'horizon infini).Par ailleurs, comme pour S et A, les fonctions de transition et de récompense peuvent

elles-mêmes varier au cours du temps, auquel cas on devrait prendre en compte ou nouvelle variable de temps. Lorsque ces fonctions de varient pas, on parle de processus stationnaires.
Les processus décisionnels de Markov permettent de modéliser la dynamique de l'état d'un système soumis au contrôle d'un agent, au sein d'un environnement stochastique. On peu ainsi définir la notion de politique (noté π). Appelé aussi stratégie, c'est la procédure suivie par l'agent pour choisir à chaque instant l'action à exécuter. On peu faire deux distinctions ici. Une politique peut déterminer précisément l'action à effectuer selon l'état ou l'on se trouve, ou simplement définir une distribution de probabilité selon laquelle cette action doit être sélectionnée. Dans notre projet on se limitera au premier cas. Il s'agit de MDP à politique déterministe. C'est le modèle le plus simple et essentiel de stratégie décisionnelle.

2.4 Critères de performance

On doit en effet ce choisir un critère afin de pouvoir juger la performance de notre politique. Cette étape est importante puisque il faut savoir choisir un critère adapté au modèle notre objectif étant de rechercher la politique qui l'optimise. En termes mathématique, cela revient à évaluer une politique sur la base d'une mesure du cumul espéré des récompenses instantanées le long de l'exécution. Voici les critères les plus étudiés au sein de la théorie des MDP:

- Le critère fini : $E[r_0 + r_1 + r_2 + \cdots + r_{N-1}|s_0]$
- Le critère -pondéré : $E[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^t r_t + \cdots |s_0]$
- Le critère total : $E[r_0 + r_1 + r_2 + \cdots + r_t + \cdots |s_0]$
- Le critère moyen : $\lim_{x\to\infty} \frac{1}{n} E[r_0 + r_1 + r_2 + \cdots + r_{n1}|s_0]$

Avec E l'espérance et r les récompenses instantanés.
Ces formules sont à la base du principe d'optimalité de Bellman [BEL 57] (≪ les sous-politiques de la politique optimale sont des sous-politiques optimales ≫).Il est à l'origine d'algorithmes de programmation dynamique qu'on verra permettant de résoudre efficacement les MDP.

2.5 Fonctions de valuation

Lorsque nous avons déterminé un critère et une politique, nous pouvons déterminé une fonction de valeur. Cette fonction associe à tout état initial $s \in S$ la valeur du critère considéré en suivant π à partir de s:

$$\forall \pi \; V^\pi \; : S \mapsto \Re$$

$V^\pi(s)$ représente le gain reçu par l'agent s'il démarre à l'état s et applique ensuite la politique π à l'infini. Les problèmes décisionnels de Markov peuvent alors être traduits en terme d'équations d'optimalité portant sur les fonctions de valeur, dont la résolution est de complexité beaucoup moins importante que le parcours exhaustif de l'espace global des politiques. C'est un vecteur avec une entré par état. Nous pouvons aussi définir une autre fonction de valuation dite états-actions : $Q^\pi(s, a)$:

$$\forall \pi \; Q^\pi \; : S \times A \mapsto \Re$$

elle représente le gain de l'agent, s'il démarre à l'état s et commence par effectuer l'action a, avant d'appliquer ensuite la politique π à l'infini. Ceci correspondrais en pratique à une matrice qui, pour chaque état possible, associe les actions possible a effectuer et leur gain.
On note que les deux fonctions sont très liées. On a en effet $V^\pi(s) = Q^\pi(s, \pi(s))$.

2.6 Équation de Belmann

Sans doute le coeur du fonctionnement des algorithmes de résolutions de MDP. La fonction de valuation vérifie une relation de récurrence:

$$V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S}^{\infty} [\, p(s'|\pi(s), s) V^\pi(s')]$$

Avec V la valeur d'être a cet état r la récompenses de prendre une action et p la probabilités de transitions.

Le but de l'agent est de trouver la politique optimale π^* qui lui permet de maximiser le gain selon le critère qu'on a choisit, c'est-à-dire la politique qui vérifie, pour tout état $s \in S$, $V^{\pi^*}(s) \geq V^\pi(s)$ pour toute autre politique π .
On peut montrer que la fonction de valeurs optimale V^* vérifie l'équation d'optimalité de Bellman:

$$V^* = \max_{a \in A} [r(s, \pi(s)) + \gamma \sum_{s' \in S}^{\infty} [\, p(s'|\pi(s), s) V^*]]$$

La fonction de valeurs optimale peut donc être approché par un algorithme itératif.

2.7 Algorithmes

Les MDP peuvent être résolus de différentes manières. Les algorithmes principale que nous allons détailler sont: Value Itération, Policy Itération et plus classiquement la Programmation linéaire.

2.7.1 Value Iteration:

La méthode itérative que nous venons de voir pour les équations d'optimalité de Bellman nous permet d'en tirer un premier algorithme d'itération sur les valeurs afin de déterminer π^*. Cet algorithme est fondé sur la programmation dynamique.
En effet si on choisit d'utiliser le critère discompté: On peu écrire la fonction de valeur ainsi:

$$\begin{aligned} V(s_0) &= r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \cdots \\ V(s_0) &= r_0 + \gamma(r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots) \\ V(s_0) &= r_0 + \gamma V(s_1) \\ \text{Et ainsi on a}: \quad V(s_t) &= r_t + \gamma V(s_{t+1}) \end{aligned}$$

Propriétés sur la récursivité:
Si on défini un opérateur T tel que: $X_{n+1} \leftarrow TX_n$
Si T est contracrif (il vaut 0.5 par exemple), alors X_n va converger après une suite d'application de T.

Ainsi on appel Operateur d'optimalité de Bellman (noté T^* souvent) l'application:

$$V_{n+1}(s) \leftarrow \max_{a \in A}[r(s, \pi(s)) + \gamma \sum_{s' \in S}^{\infty} [\, p(s'|\pi(s), s) V_n(s')]]$$

Si γ est inférieur à 1, T^* est contractif. Ainsi en itérant on obtient la valeur de la politique actuelle. La fonction de valuation optimal est donc : $V^* = T^* V^*$
En applicant T^* itérativement à la Value Fonction initial, on convergera vers la Value Fonction optimal:
$V_{n+1} \leftarrow T^* V_n$
L'algorithme en détail peut être vue dans le chapitre 1 de F.Garcia.

2.7.2 Policy Iteration:

L'algorithme d'itération sur la politique est similaire à l'algorithme précédant mais il est en deux phases: L'idée est de partir d'une politique quelconque π_0 , puis d'alterner une phase d'évaluation (dans laquelle la fonction V^{π_n} est déterminée) avec une phase d'amélioration, où l'on définit la politique suivante π_{n+1}:
Evaluation de la politique:

$$V_{n+1}^{\pi} \leftarrow T^{\pi *} V_n^{\pi}$$

Amélioration de la politique:

$$\pi'(s) \leftarrow \arg\max_{a \in A}[r(s, a) + \gamma \sum_{s' \in S}^{\infty} [\, p(s'|a, s) V_n^{\pi}(s')]]$$

Le fait de faire un évaluation a chaque itération étant très lent, un algorithme dit hybride plus efficaces consiste à faire une évaluation périodiquement. On s'arrête à convergence. L'algorithme complet est ainsi disponible dans le chapitre 1 de Garcia.

2.7.3 Programmation linéaire:

Supposons que nous ayons un modèle de MDP (transitions, récompenses, etc.).Pour tout état donné, nous avons l'hypothèse que la vraie valeur de l'état est donnée par :

$$V^*(s) = r + \gamma \max_{a \in A} \sum_{s' \in S} P(s'|s, a) \cdot V^*(s')$$

En programmation linéaire, nous trouvons la valeur minimale ou maximale d'une fonction, en respectant un ensemble de contraintes. Des solveurs programmés tel que Gurobi ou Cplex sont utilisés pour optimiser ces fonctions sous contraintes.
Nous pouvons poser le problème linéairement ainsi:

$$z = \min_V \, V(s)$$

Sous Contraintes:

$$V(s) \geq r + \gamma \sum_{s' \in S} P(s'|s,a) * V(s'),\ \forall a \in A, s \in S$$

Il est évident que si nous trouvons la plus petite valeur de V(s) qui répond à cette exigence, alors cette valeur rendra une des contraintes serrées.

2.7.4 Bilan des méthodes de résolution

Méthodes de résolution en fonction des critères				
Solveurs:	Programmation dynamique	Programmation linéaire	Value Iteration	Policy Iteration
Le critère fini	oui			
Le critère γ-pondéré		oui	oui	oui
Le critère total:				
Le critère moyen			oui	oui

3 Solveurs

Dans cette partie nous allons présentez les trois logicielles qui permettent de résoudre des MDP en discutant la disponibilité du logicielle, la facilité d'usage, les avantages et inconvénients de chacun.

3.1 MarmoteMDP

Disponibilité: MarmoteMDP est un module qui fait partie du logiciel marmoteCore disponible sur le site de son développeur Hemanuel Hyon : https://webia.lip6.fr/ hyon/Marmote/home.php. Ce logiciel propose une bibliothèque de résolution des chaînes de Markov contrôlées (Processus décisionnel de Markov).
Installation: Sur le site, le logiciel est téléchargable pour le système opérationnel Linux en langage C++ ou python. Comme nous travaillons sur une machine avec un système Windows Nous avons téléchargé une Machine Virtuelle avec le logiciel déjà installé disponible sur le même site
Fonctionnalités: MarmoteMDP offre des méthodes de résolution de modèles à horizon fini (Critère total ou escompté), ainsi que des méthodes de résolution de modèles infinis (Critère total, Critère escompté, Critère moyen). MarmoteMDP offre aussi des outils d'analyses de propriétés structurales d'un MDP.
Avantages: La modélisation d'un MDP est assez simple pour l'adapter à ce solveur. L'intégralité des méthodes pour résoudre le problème est implémenter et prêt à appliquer.
Inconvénients: Pas de résolution par programmation linéaire.

3.2 MDPToolbox

Disponibilité: MDPToolbox fournit des classes et des fonctions pour la résolution de processus de décision de Markov discret. Les classes et fonctions ont été développées sur la base de la boîte à outils MDP de MATLAB par l'Unité de Biométrie et d'Intelligence Artificielle de l'INRA Toulouse.
Installation: La boite fonctionne sous python et nécessite une installation préalable des modules Scipy et Numpy. Ensuite il y a deux manière de récuperer le package ToolBoxMDP: Depuis le Python Packaging Index(pip) ou en faisant un clone du répository github.
Fonctionnalités: La liste des algorithmes qui ont été implémentés comprend la programmation linéaire, l'itération de politique, le q-learning et l'itération de valeur ainsi que plusieurs variations.
Avantages: Ce logiciel dispose la bibliothèque d'algorithmes la plus complète. Une documentation très riche en définition et exemple d'utilisation des méthodes implémentés est aussi disponible afin de faciliter l'utilisation du logiciel.
Inconvénients: Développé uniquement pour le langage Pyhton.

3.3 Gurobi

Disponible: Le solveur Gurobi est disponible sur le site officiel: https://www.gurobi.com/products/gurobi-optimizer/.
Installation: Il est payant pour les entreprises qui souhaite l'utiliser en industrie, mais une licence temporaire gratuite peut être obtenu pour les étudiants souhaitant s'entraîner ou faire des recherche. Il peut être téléchargé ensuite et activé sous forme de module Python qu'on importe dans un éditeur comme spider, afin d'utiliser les fonctionnalités implémentés. Python est le langage de notre choix mais des versions pour d'autre langage comme C++, Java ou R sont aussi disponibles.
Fonctionnalités: Après la création d'un modèle de programmation mathématique Gurobi offre une interface de ligne de commande pour calculer une solution optimale.
Avantages: La résolution d'un MDP sous forme de programme linéaire offre le plus de liberté.

Inconvénients: Difficile de modéliser un MDP en programme linéaire.

3.4 Comparaison des solveurs

4 Exemples de Problèmes de Décision à résoudre