Etude comparative de solveurs pour les Processus de Décisions Markovien

Calic Petar
Sorbonne Université, Paris
Encadré par Emmanuel Hyon

petarcalic99@gmail.com

26/05/21

Résumé

Nous avons eu dans ce projet comme objectif, de faire une comparaison numérique de solveurs pour des MDP. La première étape a été constituée de recherches sur les fondements des MDP pour les maîtriser. Essentielles à la maîtrise et à l'utilisation des trois solveurs étudiés dans ce projet. La deuxième étape a été ensuite de développer un benchmark, auquel ont été soumis les solveurs.

Mots-clés: MDP; Solveur; Benchmark

Note: Muyang Shi a malheureusement quitté l'équipe du projet début mars. Ceci a ralenti l'avancement du projet mais nous avons adapté la phase finale du projet afin d'avoir des résultats intéressants. La partie de construction de benchmark pour une analyse complète ne sera donc pas présentée dans le rapport mais une comparaison de plusieurs exemples simple de MDP pour chaque cas que l'on peu rencontrer.

1 Introduction

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

1.1 Objectifs du Projet:

1.1.1 Maitriser les MDP

Le but de ce projet est de comparer plusieurs méthodes de résolution de MDP et des solveurs de 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 doit être vérifiée. 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) MarmoteMDP et MDPToolbox. 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. Elle fait des Itérations sur les valeurs pour chaque état 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 du MDP en un problème linéaire et son Dual pour le résoudre.

Les MDP peuvent avoir plusieurs critères d'optimisation et selon ces derniers, les algorithmes ne sont pas les mêmes à utiliser. Les types de résultats (politiques) ne sont aussi pas les mêmes.

1.1.3 Modéliser un problème MDP pour solveur

Pour pouvoir comparer les solveurs, nous allons prendre un ensemble d'exemples de problèmes mobilisable 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 informatique pour un solveur spécifique. En effet, chaque solveur requiert une modélisation informatique 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 exécuter 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 trois mois et demi de travail. Ceci est notre estimation de temps nécessaire pour faire un travail de qualité sans grande pression du temps.
L'ensemble du projet sera codé sous Python à l'aide d'une machine virtuelle sous Linux(notamment à cause de la portabilité limitée de l'un des solveurs). Le rapport sera 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, cela peut être vu 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 ceci correspond à 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'étape antérieure dans l'état s;
- une fonction de récompense $\Re : S \times A \times S \to \mathbb{R}$; elle définit la récompense (positive ou négative) reçue par l'agent:$R(s, a, s')$ est la récompense 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 :
  - $\Re : S \times A \times S \to \mathbb{R}$ (récompense déterministe, c'est le choix que nous adopterons dans la suite.)
  - $\Re : S \times \mathrm{A} \to \mathbb{R}$ (récompense déterministe rattachée à l'action en ignorant son résultat)

2.1.1 Illustration d'un MDP:

Soit $s0$ l'état initial d'un agent. On choisit dans A une action $a0$ à exécuter. L'agent passe aléatoirement à l'état suivant $s1$ en fonction de la probabilité de T. Ensuite, l'action a1 permet de passer de s1 à s2 en fonction de T, et ainsi de suite. Ce processus est représenté sur 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.

2.2 Un exemple 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.

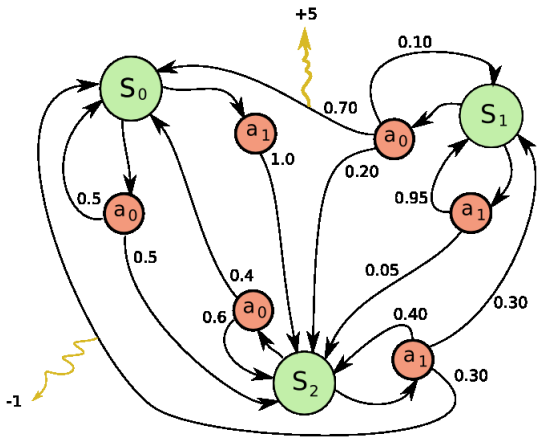

Figure 2: Exemple d'un MDP.

La matrice de transition associée à l'action a_0 est la suivante :

$$p_0 = \begin{pmatrix} 0.50 & 0 & 0.50 \\ 0.70 & 0.10 & 0.20 \\ 0.40 & 0 & 0.60 \end{pmatrix}$$

La matrice de transition associée à l'action a_1 est la suivante :

$$p_1 = \begin{pmatrix} 0.50 & 0 & 0.50 \\ 0.70 & 0.10 & 0.20 \\ 0.40 & 0 & 0.60 \end{pmatrix}$$

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 N des étapes de décisions est un ensemble discret, qui peut être fini ou infini (on parle d'horizon fini ou d'horizon infini). Par ailleurs les fonctions de transition et de récompense peuvent elles-mêmes varier au cours du temps. Dans ce cas, une nouvelle variable de temps devrait être prise en compte. Lorsque ces fonctions ne 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 peut ainsi définir *la notion de politique (notée π)*, appelée aussi stratégie, comme la procédure suivie par l'agent pour choisir à chaque instant l'action à exécuter. Deux distinctions peuvent donc être faites. Soit une politique peut déterminer précisément l'action à effectuer selon l'état où l'on se trouve, soit définir une distribution de probabilité selon laquelle cette action doit être sélectionnée. Ce projet se limitera au premier cas : Il s'agit de MDP à *politique déterministe.* C'est le modèle le le mieux adapté à notre niveau de connaissance, qui est aussi optimal comme stratégie décisionnelle.

2.4 Critères de performance

Afin de définir un MDP et de juger de la performance de la politique sélectionnée, un critère de performance doit être choisi (L'objectif étant de chercher la politique qui optimise le modèle, il faut savoir choisir un critère qui y est le mieux adapté). En termes mathématiques, 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 instantanées.

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 ≫) qui est à l'origine d'algorithmes de programmation dynamique permettant de résoudre efficacement les MDP.

2.5 Fonctions de valuation

Une fois un critère et une politique choisis, une fonction de valeur peut être déterminée. Cette fonction associe à tout état initial $s \in S$ la valeur du critère considéré en suivant π à partir de s:

$$\forall \pi \; V^\pi \; : S \mapsto \mathbb{R}$$

$V^\pi(s)$ 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 entrée par état. Nous pouvons aussi définir une autre fonction de valuation dite états-actions : $Q^\pi(s,a)$:

$$\forall \pi \; Q^\pi \; : S \times A \mapsto \mathbb{R}$$

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 correspondrait en pratique à une matrice qui, pour chaque état possible, associe les actions possibles à 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

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

C'est sans doute le coeur du fonctionnement des algorithmes de résolutions de MDP. La fonction de valuation vérifie une relation de récurrence:

$$V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S}^{\infty} [\, T(s'|\pi(s), s) V^\pi(s')]$$

Ceci correspond à l'opérateur pour le critère escompté, il faut légèrement adapter la formule selon le critère. Avec V la valeur d'être dans cet état et r la récompense de prendre une action et T la probabilité de transition.

Le but de l'agent est de trouver la politique optimale π^* qui lui permet de maximiser le gain selon le critère choisi, 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:

$$V^* = \max_{a \in A} [r(s, \pi(s)) + \gamma \sum_{s' \in S}^{\infty} [\, T(s'|\pi(s), s) V^*]]$$

La fonction de valeur peut donc être approchée par un algorithme itératif.
En effet si on choisit d'utiliser le critère escompté, la fonction de valeur peut être ainsi écrite :

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

2.7 Algorithmes

Les MDP peuvent être résolus de différentes manières. Les algorithmes principaux qui seront détaillés par la suite: Value Iteration, Policy Iteration et plus classiquement la Programmation linéaire.

2.7.1 Value Iteration:

La méthode itérative, présentée dans la partie précédente pour les équations pour les équations d'optimalité de Bellman, permet d'élaborer un premier algorithme d'itération sur les valeurs afin de déterminer à la fois V^* et π^*. Cet algorithme est fondé sur la programmation dynamique.

On appelle ainsi 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 contractant. En appliquant T^* itérativement à la Value Fonction initiale, on aura convergence vers la Value Fonction optimale:

$$V_0 \leftarrow 0$$
$$V_{n+1} \leftarrow T^* V_n$$

Ainsi on doit poser un ϵ très petit:

$$|V_{n+1} - V_n| \leq \epsilon$$

Pour obtenir $V^* = V_n$.

Ensuite on en déduit la politique ainsi:

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

L'algorithme en détail peut être vu dans le chapitre 1 de F.Garcia.

2.7.2 Policy Iteration:

L'algorithme d'itération sur la politique est similaire à l'algorithme précédent mais il est construit en deux temps. L'idée est de partir d'une politique quelconque π_0 , puis d'alterner une phase d'évaluation (durant laquelle la fonction V^π est évaluée) avec une phase d'amélioration, où l'on définit la politique suivante π_{n+1}:
$\pi_0 = 0$
$V_0 = 0$

$$\pi_0(s) \leftarrow \arg\max_{a \in A} [r(s,a) + \gamma \sum_{s' \in S}^{\infty} [\, T(s'|a,s) V_0^\pi(s')]]$$

Évaluation de la politique:

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

Amélioration de la politique:

$$\pi'(s) \leftarrow \arg\max_{a \in A} [r(s,a) + \gamma \sum_{s' \in S}^{\infty} [\, T(s'|a,s) V_n^\pi(s')]]$$

Faire une évaluation à chaque itération étant très lent, un algorithme "hybride" plus efficace 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 faisons 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 exprimant l'opérateur de Programmation Dynamique sous la forme d'un programme linéaire satisfaisant un ensemble de contraintes. Des solveurs programmés tels que Gurobi ou Cplex sont utilisés pour optimiser ces fonctions sous contraintes.
Nous pouvons ainsi poser le problème linéairement (ici c'est un critère escompté):

$$z = \min_V \frac{1}{|S|} \sum_s V(s)$$

Sous Contraintes:

$$V(s) \geq r + \gamma \sum_{s' \in S} T(s'|s,a) * V(s'),\ \forall a \in A, s \in S$$

Si n et m sont les tailles respectives de S et A, avec $p()$ et $r()$ codées sur b bits, la complexité d'un tel algorithme de programmation linéaire sur les rationnels est polynomiale en n, m, b avec des temps de résolution assez lents. Nous verrons toutefois qu'il existe des méthodes de programmation linéaire qui peuvent s'avérer très efficaces dans le cadre des MDP admettant une représentation factorisée.

2.7.4 Bilan des méthodes de résolution

Les types d'algorithme qui peuvent être utilisés sont détaillés ci-dessous pour chacun des critères.

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

"oui*" signifie qu'une version modifiée de l'algorithme doit être utilisée. Ce tableau à été résumé à partir du chapitre 1 du livre de F. Garcia. et du livre de M. Puterman(en bleu).

3 Solveurs

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

3.1 MarmoteMDP

MarmoteMDP est un module qui fait partie du logiciel marmoteCore disponible sur le site de son développeur Emmanuel 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).
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 télechargé une Machine Virtuelle avec le logiciel déjà installé (disponible sur le même site).
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). Il offre aussi des outils d'analyses de propriétés structurales d'un MDP.
La modélisation d'un MDP est assez simple et directe pour l'adapter à ce solveur. Une utilisation très agréable pour les utilisateurs non habitués à la programmation. L'intégralité des méthodes pour résoudre les problèmes de décision sont implémentées et prêtes à appliquer. Bien que ce logiciel soit très efficace, en terme de temps de calcul, le logiciel retourne aussi beaucoup d'informations supplémentaires liées à la résolution de l'MDP.
En revanche, MarmoteMDP ne permet toujours pas la résolution par programmation linéaire. Un autre inconvénient est que le logiciel n'est disponible que sur Linux pour l'instant et l'utilisation de la machine virtuelle est assez contraignante.

3.2 MDPToolbox

MDPToolbox fournit des classes et des fonctions pour la résolution de processus de décision de Markov discrets. 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.
Le module fonctionne sous Python et nécessite une installation préalable des modules Scipy et Numpy. Ensuite, il y a deux manières de récupérer le package ToolBoxMDP: Depuis le Python Packaging Index(pip) ou en faisant un clone du répository Github.
La liste des algorithmes qui ont été implémentés comprend la programmation linéaire, l'itération de politique, le q-learning et l'itération sur les valeurs ainsi que plusieurs variations.

Ce logiciel dispose donc d'une bibliothèque d'algorithmes assez complète, et simple à utiliser. Une documentation avec des définitions et exemples d'utilisations des méthodes implémentées est aussi disponible afin de faciliter l'utilisation du logiciel. Voici le lien du site avec la documentation du module: https://pymdptoolbox.readthedocs.io/en/latest/api/mdp.html
Il a ses inconvénients : Il est développé uniquement pour le langage Python et il permet aussi de construire des MDP uniquement à critère escompté. Une autre lacune est qu'il ne peu résoudre que des problèmes de maximisation.

3.3 Gurobi

Le solveur de programme linéaire Gurobi est disponible sur le site officiel: https://www.gurobi.com/products/gurobi-optimizer/.
Il est payant pour les entreprises qui souhaitent l'utiliser en industrie, mais une licence temporaire gratuite peut être obtenue pour les étudiants et professeurs souhaitant l'utiliser pour des fin éducatives et pédagogique. Il peut être téléchargé ensuite et activé sous forme de module Python qu'on importe dans un éditeur comme Spyder, afin d'utiliser les fonctionnalités implémentées. Python est le langage choisi dans le cadre de ce projet mais des versions pour d'autre langage comme C++, Java ou R sont aussi disponibles.
Après la création d'un modèle de programmation mathématique, Gurobi offre une interface de ligne de commande pour calculer une solution optimale.
Gurobi offre le plus de liberté pour la résolution d'un MDP sous forme de programme linéaire.
La programmation linéaire ne s'est pas montrée jusqu'à présent comment étant une méthode efficace de résolution de MDP. Néanmoins, des innovations à venir pourront changer cela (selon M.Puterman). Il n'est souvent pas évident de représenter un MDP sous forme de programme linéaire qui sera résolu par Gurobi. Aussi, son utilisation pour la résolution de MDP nécessite d'être familier en particulier avec Gurobi.

3.4 Comparaison des solveurs

Chacun des trois logiciels sera utilisé selon le problème et l'analyse souhaitée, tout en tenant compte des avantages et des inconvénients de chacun. MDPToolBox dispose de l'interface la plus simple à utiliser (le moins de ligne de code a entrer). Néanmoins, la modélisation est assez lourde à faire puisque chaque matrice doit être définie sous forme de tableau. Ces matrices peuvent être très grandes et creuses (peuvent contenir beaucoup de valeurs nul). MarmoteMDP à l'opposé demande uniquement de renseigner les valeurs non nulles des matrices, ce qui rend la tâche de modélisation beaucoup plus légère et rend la construction du modèle par le solveur plus rapide. MDPToolBox aussi offre peu de liberté dans l'organisation et la manipulation des calculs. L'utilisateur n'a par exemple pas de moyen d'agir sur les contraintes du programme linéaire et fonctionne plutôt sous le principe de boite noire. Le solveurs aussi ne peu résoudre que des problèmes de maximisation.
Les résultats retournés sont beaucoup moins détaillés que chez MarmoteMDP. Ce dernier serait sans doute le solveur le plus adapté pour une étude plus analytique sur les MDP, puisque le programme renvoie, avec les résultats du problème, une multitude de données pertinentes pour une analyse numérique. Pour une résolution par programmation linéaire, l'option plus rapide est sans doute d'utiliser MDPToolBox qui a des méthodes prédéfinies, mais pour une étude analytique il faudrait utiliser Gurobi qui est spécifiquement un solveur de programme linéaire et requiert que les programmes soient modélisés manuellement.

3.5 Plateforme de comparaison

Afin de comparer les différentes exécutions des problèmes avec des solveurs différents, nous avons mis en place une plateforme de comparaison. En effet, nous avons utilisé la Machine Virtuelle Linux et nous avons installé sur cette dernière les modules et licences nécessaires afin de pouvoir lancer les problèmes sur des environnements où les temps d'exécutions sont comparables. Cette étape nécessitait d'apprendre à utiliser et à manipuler des Machines Virtuelles puisque plusieurs problèmes étaient survenus tels qu'une mémoire et un espace disque insuffisants.

4 Quelques problèmes de Décision

4.1 Critère escompté

4.1.1 Modélisation - Exemple1

Voici un premier exemple très simple de modèle infini et critère escompté. On observe un modèle à deux états (x_1, x_2) et deux actions possibles: (a_1, a_2). Voici les matrices de transition et de récompense: On représente chacune des matrices associées à une action. P_1 et P_2, associées aux décisions 1 et 2 respectivement.

$$P_1 = \begin{pmatrix} 0.6 & 0.4 \\ 0.5 & 0.5 \end{pmatrix} \text{ et } P_2 = \begin{pmatrix} 0.2 & 0.8 \\ 0.7 & 0.3 \end{pmatrix}$$

Par exemple, 0.6 correspond à la probabilité de rester à l'état x_1 sachant qu'on était à l'état x_1 et qu'on a fait l'action a_1. Elle devient 0.2 si on effectue l'action a_2. 0.4 est la probabilité de venir à l'état x_2 sachant qu'on était à l'état x_1 et qu'on a fait l'action a_1.
La matrice de récompense est représentée ainsi:

$$R = \begin{pmatrix} 4.5 & 2 \\ -1.5 & 3.0 \end{pmatrix}$$

Les lignes représentent les états et les colonnes les actions possibles.
Par exemple 4.5 est le gain obtenu en faisant l'action a_1 à l'état x_1.
Ce type de modèle peut être résolu à l'aide le Value Iteration, de Policy Iteration et de la programmation linéaire.

4.1.2 Code

L'ensemble du code est mis en annexe. Néanmoins, évoquons quelques points importants:

- MarmoteMDP: La résolution d'un MDP peut être divisée en deux étapes principales dans MarmoteMDP. On commence par construire le modèle. On crée les matrices de transitions et de coût. On initialise aussi toutes les constantes nécessaires. A l'aide de la méthode MDP1 = discountedMDP(critere, stateSpace, actionSpace, trans, Reward,beta) on a défini notre MDP avec le critère escompté. La deuxième étape est celle de la résolution de l'MDP avec la commande mdp1.valueIteration(epsilon, maxIter) pour une résolution avec l'algorithme Value Iteration. Le solveur offre, en plus de la politique optimale, beaucoup d'informations supplémentaires sur la résolution, comme le nombre d'itérations et la valeur de chaque état calculé. Nous avons rajouté simplement des commandes de chronométrage pour calculer les temps d'exécution.

- MDPToolbox: Ce solveur a une structure très similaire à MarmoteMDP et se déroule en deux étapes principales : la construction du modèle, suivie de la résolution de ce dernier. Contrairement à MarmoteMDP, la construction du modèle ne se fait pas par rapport au critère mathématique(Horizon infini,critère escompté) mais par rapport à l'algorithme que l'on souhaite utiliser (Value Iteration) et le solveur sous entend qu'il s agit du critère escompté. Voici la méthode: vi = mdptoolbox.mdp.ValueIteration(). Ceci laisse beaucoup moins de liberté pour la modélisation de MDP. De plus, la méthode de résolution (vi.run()) ne renvoie rien. L'objet "vi" est surchargé et il faut appeler tout élément dont on a besoin séparément.

- Gurobi: Gurobi demande d'avoir des pré-requis en programmation linéaire ; il faut donc savoir modéliser le problème avant de pouvoir utiliser ce solveur. Cependant, la résolution d'un problème reste assez élégante une fois ces points acquis. Tout d'abord, il faut définir toute les matrices et paramètres du problème. En suite poser le programme linéaire: Il faut définir les variables de décisions (Les valeurs v(s)) et en suite définir la fonction objectif à minimiser (somme des v(s)). La partie clé du programme en suite est la définition des contraintes sur la fonction à minimiser.

  Voici le modèle:

$$z = \min_V \frac{1}{n}(v_1 + v_2)$$

$$s.c.\ v_1 \geq R[1,0] + \gamma \cdot P[1][1,0] \cdot v_1 + \gamma \cdot P[2|1,0] \cdot v_2$$
$$v_1 \geq R[1,1] + \gamma \cdot P[1|1,1] \cdot v_1 + \gamma \cdot P[2|1,1] \cdot v_2$$
$$v_2 \geq R[2,0] + \gamma \cdot P[2|2,0] \cdot v_1 + \gamma \cdot P[2|2,0] \cdot v_2$$
$$v_2 \geq R[2,1] + \gamma \cdot P[2|2,1] \cdot v_1 + \gamma \cdot P[2|2,1] \cdot v_2$$
$$v_1, v_2 \in \mathbb{R}$$

4.2 Critère moyen

Exemple2
Ce modèle correspond à l'exemple 8.5.3 du livre de Puterman. Même s'il reste simple, il s'agit d'un exemple très pratique et important puisqu'il a été calculé formellement. Ayant obtenu des résultats différents de fonctions d'évaluation pour le premier exemple nous avons choisi cet exemple pour vérifier que les algorithmes fonctionnent correctement.
On a un espace d'état de taille 3 (s_0, s_1, s_2) et 4 actions possibles : deux dans l'état $s_0 : a_0, a_1$. Une dans l'état $s_1 : a_2$ et une dernière dans l'état $s_2 : a_3$.
Voici les matrices de transitions et de récompenses:

$$P_0 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \ et\ P_1 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \ et\ P_2 = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \ et\ P_3 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{pmatrix}$$

L'ensemble d'actions possibles change en fonction de l'état. On va supposer que l'espace d'action a toujours la même taille quelque soit l'état. Ceci implique que les actions impossibles dans un état ne doivent pas avoir d'effets (ce qui se traduit par une transition vers soit même). Donc, elles doivent être pénalisées pour éviter qu'elle soient choisies:

$$R = \begin{pmatrix} 2 & 1 & -\infty & -\infty \\ -\infty & -\infty & 2 & -\infty \\ -\infty & -\infty & -\infty & 3 \end{pmatrix}$$

4.2.1 Code

- MarmoteMDP: Comme dans l'exemple précédent, on commence par définir les matrices. Les lignes des matrices de transitions doivent se sommer a 1. Aussi On ne doit pas rentrer de valeurs nuls dans les matrices de transition dans MarmoteMDP. Dans la matrice des récompenses pour les "$-\infty$" on met des "-1000". Ensuite, on initialise le problème à l'aide d'une méthode: averageMDP(critere, stateSpace, actionSpace, trans, Reward). L'étape de résolution se déroule en choisissant l'algorithme comme ceci: mdp1.valueIteration(epsilon, maxIter).

- MDPToolbox: Ce solveur ne permet pas de modéliser un problème selon le critère moyen. Nous devons donc rendre les matrices stochastiques, puis construire le modèle selon avec la méthode RelativeValueIteration().

- Gurobi: Le programme linéaire à résoudre pour ce critère est différent. Si n est la taille de l'espace d'état il y a n+1 variables de décision pour ce PL: Le scalaire g et $h(s)$ avec s les n etats. $h(s)$ correspond à $v(s)$ dans les cas précedant. g est le terme de la valeure moyenne. Pour résoudre le système, on programme donc:

  $$z = \min g$$

  Sous Contraintes:
  $$h(s) >= r(s,a) - g + \sum_{s'} p(s'|s,a)h(s'),\ \forall a \in A, s \in S$$

4.3 Critère total

4.3.1 Horizon infini

Exemple3
Nous allons à travers cet exemple tester la performance des algorithmes sur un modèle à horizon infini et à critère total. Nous n'avons pas de modèle particulier relié à cet exemple. On va supposer que les matrices sont :

$$R = \begin{pmatrix} 0 & 4000 & 6000 \\ 1000 & 4000 & 6000 \\ 3000 & 4000 & 6000 \\ 3000 & 4000 & 6000 \end{pmatrix}$$

Et les matrices de transition:

$$P_0 = \begin{pmatrix} 0 & 0.875 & 0.0625 & 0.0625 \\ 0 & 0.75 & 0 & 0.25 \\ 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 1 \end{pmatrix} \; et \; P_1 = \begin{pmatrix} 0.875 & 0 & 0.125 & 0 \\ 0 & 0.75 & 0.125 & 0.125 \\ 0.8 & 0 & 0.2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$P_2 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Exemple31
Il s'agit d'un modèle de *plus court chemin* stochastique donné dans le magazine "Tangente" consacré à la Recherche Opérationnelle, dans lequel un élève doit se rendre d'un état "maison" à un état "collège".

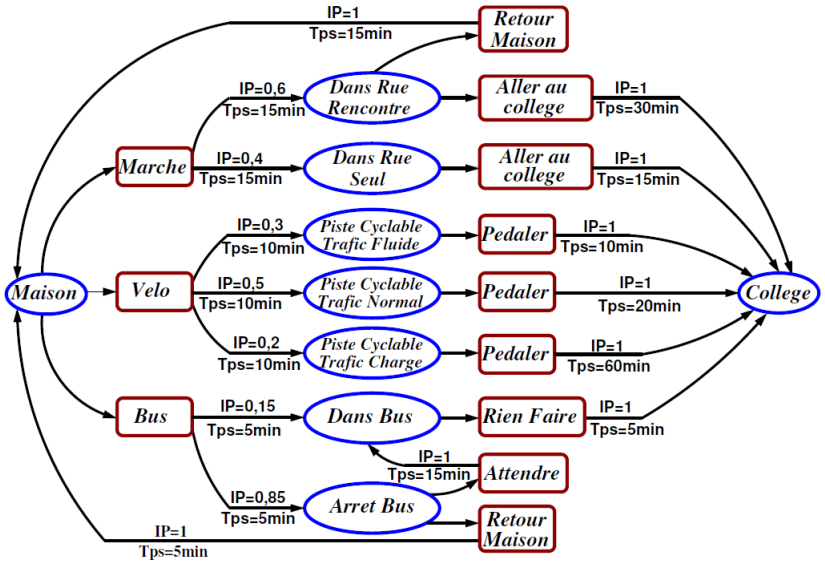

Figure 3: schéma du problème

Les états sont représentés en bleu et les actions en bleu.
Il se décompose donc ainsi: L'état "Maison" va être représenté par 0 ; l'état "Rue Rencontre" va être représenté par 1 ; l'état "Rue Seul" va être représenté par 2; et ainsi de suite. Similairement, l'action "Marche" va être représentée par 0 ; l'action "Vélo" va être représentée par 1 ; l'action "Bus" va être

représentée par 2 ; Et ainsi de suite. On va modéliser les récompenses ici par le temps que prend le déplacement que l'on veut minimiser. Les probabilités sont indiquées sur le schéma.
On peut donc représenter ces données sous forme matricielle.
Voici la matrice de récompenses (notons que les valeurs creuses sont représentées par des $+\infty$ puisqu'on est dans un problème de minimisation):

$$R = \begin{pmatrix} 15 & 10 & 5 & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 10 & 30 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 15 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 10 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 20 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 60 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 5 & \infty \\ \infty & \infty & \infty & 5 & \infty & \infty & \infty & 15 \\ \infty & \infty & \infty & 0 & \infty & \infty & 0 & \infty \end{pmatrix}$$

Voici les matrices de transitions pour les actions 0 (marcher) et 1(velo):

$$P_0 = \begin{pmatrix} 0 & 0.6 & 0.4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \text{ et } P_1 = \begin{pmatrix} 0 & 0 & 0 & 0.3 & 0.5 & 0.2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Code
Pour MarmoteMDP, la manipulation informatique va utiliser un objet de type totalRewardMDP. Le nombre d'itération va être relié au nombre d'étapes du chemin. Pour la construction de la matrice des récompenses, on met de grandes valeurs pour simuler la valeur infini qui sert à faire en sorte que les actions associés ne soit jamais sélectionnées. MDPToolBox ne permet pas de faire des minimisation et on ne peu pas traiter le problème avec ce solveur. Concernant la programmation linéaire pour passer au problème de minimisation, il faut donc penser à faire un minimisation de la fonction objectif, et inverser le sens des contraintes.

4.3.2 Horizon fini

Exemple4
Il s'agit d'un modèle à horizon fini et à critère total. Il s'agit du modèle dit "stochastic inventory model" étudié dans le livre de Puterman.
C'est un problème de gestion de stock sur un horizon de temps fini $N = 4$. On a un espace d'état de taille $M = 4$, l'espace d'action est positif et aussi de taille 4. La matrice de revenu est :

$$R = \begin{pmatrix} 0 & -1 & -2 & -5 \\ 5 & 0 & -3 & -\infty \\ 6 & -1 & -\infty & -\infty \\ 5 & -\infty & -\infty & -\infty \end{pmatrix}$$

Voici les matrices de transition:

$$P_0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.75 & 0.25 & 0 & 0 \\ 0.25 & 0.5 & 0.25 & 0 \\ 0 & 0.25 & 0.5 & 0.25 \end{pmatrix} \text{ et } P_1 = \begin{pmatrix} 0.75 & 0.25 & 0 & 0 \\ 0.25 & 0.5 & 0.25 & 0 \\ 0 & 0.25 & 0.5 & 0.25 \\ 0 & 0.25 & 0.5 & 0.25 \end{pmatrix}$$

$$P_2 = \begin{pmatrix} 0.25 & 0.5 & 0.25 & 0 \\ 0 & 0.25 & 0.5 & 0.25 \\ 0 & 0.25 & 0.5 & 0.25 \\ 0 & 0.25 & 0.5 & 0.25 \end{pmatrix} \; et \; P_3 = \begin{pmatrix} 0 & 0.25 & 0.5 & 0.25 \\ 0 & 0.25 & 0.5 & 0.25 \\ 0 & 0.25 & 0.5 & 0.25 \\ 0 & 0.25 & 0.5 & 0.25 \end{pmatrix}$$

4.3.3 Code

- On manipule ici des MDP à horizon fini et critère total. Le seul algorithme qui résout ce genre de modèle est Value Iteration. Il faut faire attention au différent fonctionnement des deux solveurs. En effet MDPToolBox ne construit pas de modèle à critère total, mais construits un objet *finiteHorizonMDP()* qui utilise le critère escompté. La valeur par défaut de γ est 0.5, il faut donc passer 1 en paramètre pour parvenir à l'équivalent d'un critère total. Il peut être problématique de ne pas être conscient de ce détail non mentionné dans la documentation. MarmoteMDP au contraire permet de construire un modèle specifiquement à crtère total. De plus il n'y a pas de moyen de vérifier l'exactitude de la solution'à part de résoudre le problème formellement. Ce MDP est résolu dans le livre de M. Putemran à la page 95. Nous avons ainsi pu mettre en évidence le problème avec le solveur MDPToolBox.
- Programmation linéaire: On s'aperçoit que résoudre linéairement un MDP à critère total à Horizon infini revient à résoudre un MDP à critère escompté avec un $\gamma = 1$.

5 Résultats numériques

Dans l'ensemble des exemples, pour les temps de calcul (noté Temps Cal), et de construction des modèles (Temps Mod) on a fait une moyenne sur 10 exécutions, en secondes.

5.1 Critère escompté

Exemple1:
Résultats Attendus $V(0) = 8,4285$ et $V(1) = 6.9998$

On pose $\gamma = 0.5$, un nombre de pas maximal à 700 et un $\epsilon = 0.0001$

Exemple1:				
Algorithme:	V(S)	Temps Cal	Itérations	Temps Mod
MarmoteMDP-V.I.	(8.428541,6.999970)	4.876e-05	18	3.266e-05
MarmoteMDP-P.I.M	(8.428555 ,6.999983)	4.132e-05	3	3.266e-05
MarmoteMDP-V.I.GS	(8.428556 ,6.999991)	2.410e-05	15	3.266e-05
MDPToolbox-V.I.	(8.181412, 6.752840)	0.00023	5	0.000524
MDPToolbox-P.I.	(8.428571, 7.000000)	0.0036	1	0.000434
MDPToolbox-V.I.GS	(8.420540, 6.995569)	0.00033	7	0.000321
MDPToolbox-PL	(8.428571, 6.999999)	0.028	0	0.000304
Gurobi-PL	(8.42857,7.000000)	0.02	0	0.025

Conclusion: Même s'il s'agit d'un exemple très petit, et qu'on ne peut pas tirer de conclusion définitive, on a bien certaines indications sur le comportement des solveurs. MarmoteMDP semble être le plus précis et s'approche le plus de la valeur théorique attendue. C'est aussi le plus consistent. MDPToolbox a de bons résultats pour Policy Iteration et Gauss Seidel Value Iteration, mais on peut considérer que la précision de Value Iterations n'est pas satisfaisante si l'on prend en compte la taille du modèle. Il faudrait augmenter le nombre d'itération et diminuer ϵ. Policy Iteration hybride et Value Iteration Gauss Seidel sont les algorithmes les plus efficaces et performants pour ce type de problème. Le solveur le plus rapide est MarmoteMDP. D'autant plus que MarmoteMDP a besoin de faire qu'une seule construction de modèle selon le critère choisi et fait la résolution avec le choix d'un algorithme dans une autre étape. MDPToolBox doit construire le modèle pour chaque type d'algorithme que l'on souhaite utiliser. MarmoteMDP serait

donc beaucoup plus adapté pour un grand problème où l'on souhaite exécuter plusieurs algorithmes différents.

5.2 Critère moyen

Exemple2
Résultats Attendus: On doit obtenir 2.5 comme coût moyen. Ce résultat est calculé formellement dans le livre de M.Puterman. Il s'agit d'un modèle construit spécifiquement pour tester la bonne programmation du solveur puisque c'est un modèle qui ne converge pas. Il est fait pour montrer qu'il y a des risques de divergence des méthodes de résolutions notamment si les matrices de transition ne sont pas stochastiques. C'est la raison pour laquelle l'algorithme a fait 1000 itérations et s'éloigne de la solution théorique: On fixe un nombre de pas maximal à 1000 et un $\epsilon = 0.0001$

Exemple2:				
Algorithme:	V(S) *moyen*	Temps Cal	Itérations	Temps Mod
MarmoteMDP-V.I.	2.378235	0.000658	1000	2.813e-0.5
MarmoteMDP-P.I.M	2.260885	0.000741	1000	2.813e-0.5
MDPToolBox-V.I.	x	x	x	x

Le problème ce situe dans la troisième matrice de transition. On a trois valeurs $\frac{1}{3}$ dont la somme ne fait pas 1 mais tend vers 1. On doit modifier un élément à 0.3334 par exemple pour que la somme soit bien 1 et la matrice bien stochastique:

Exemple2 corrigé:				
Algorithme:	V(S) *moyen*	Temps Cal	Itérations	Temps Mod
MarmoteMDP-V.I.	2.500038	4.908621e-05	20	4.24385e-0.5
MarmoteMDP-P.I.M	2.500023	1.940654e-05	5	3.05175e-0.5
*MarmoteMDP-R.V.I	2.500023	4.908621e-05	20	3.74317e-0.5
*MDPToolBox-R.V.I.	2.497985	0.00081610	11	0.00028995
Gurobi-PL	2.500025	0.05	2	0.03

*On peut utiliser l'algorithme Relative Value Iteration.

Conclusion: Cet exemple de MDP irrégulier nous a permis de voir le comportement des solveurs avec des matrices de transition non stochastiques, ainsi que la procédure à suivre dans ce cas. MDPToolBox renvoie une erreur si les matrices ne sont pas stochastiques alors que Marmote effectue des calculs mais sans convergence. Apres ajustement des matrices pour les rendre stochastique nous avons pue utiliser le critère moyen et trouver la solution du problème avec MarmoteMDP. MDPToolBox ne travail qu'avec le critère escompté et on peut donc calculer la solution qu'avec l'algorithme Relative Value Iteration.

5.3 Critère Total

Exemple4: Horizon fini
Nous avons traité ici un exemple de MDP à horizon fini. Il se fait donc en N étapes que l'on a résumés dans le tableau suivant:
On pose un nombre de pas maximal à 1000 et un $\epsilon = 0.0001$ bien que $N = 4$

MarmoteMDP Value Iteration				
Step	1	2	3	4
V(0)	6.625	4.1875	2	0
V(1)	10.156	8.0625	6.25	5
V(2)	14.109	12.125	10	6
V(3)	16.625	14.188	10.5	5

MarmoteMDP calcule la solution avec un temps moyen de 3.02791e-05 secondes.

MDPToolBox - FiniteHorizon				
Step	1	2	3	4
V(0)	6.625	4.1875	2.	0
V(1)	10.15625	8.0625	6.25	5
V(2)	14.109375	12.125	10.	6
V(3)	16.625	14.1875	10.5	5

MarmoteMDP calcule la solution avec un temps moyen de 0.000272 secondes.
Nous pouvons aussi essayer de augmenter N. Ainsi avec N = 40, MDPToolBox trouve la solution en 0.001387 secondes. MarmoteMDP est plus rapide et calcule les 40 étapes en 7.177757e-05 secondes.
Conclusion: Les deux solveurs renvoient des résultats qui correspondent aux résultats calculés formellement dans le livre de Puterman. MarmoteMDP est pour ce modèle aussi plus rapide.

Exemple3: Horizon infini
Nous avons découvert que le modèle que nous avons construit ne converge pas même si les matrices sont stochastiques. Donc un modèle lui même peut ne pas être convergent due aux valeurs des matrices des gains. On voit ainsi que MarmoteMDP fait MaxIter itérations.

Exemple3:				
Algorithme:	V(S)	Temps Cal	Itérations	Temps Mod
MarmoteMDP-V.I.	(29.8e6, 29.8e6, 2.99e6, 3e6)	0.00078	1000	2.425e-0.5
MDPToolBox-V.I.	(6000, 6000, 6000, 6000)	8.885e-0.5	1	5.55e-0.5
Gurobi	(29.99e6, 29.9e6, 2.99e6, 3e6)	0.02	2	0.03

Nous allons donc trouver un problème pour ce critère calculé formellement afin d'être sûr que les algorithmes convergent et ainsi avoir des résultats pertinents pouvant être comparés.

Exemple31: Horizon infini
Résultat attendu: V(S) = (22.1125, 27.5,15.00,10.00,20.00,60.00,5.00,20.00,0.00)

On pose un nombre de pas maximal à 1000 et un $\epsilon = 0.0001$

Exemple31:				
Algorithme:	V(S)	T Cal	Itérations	T Mod
MarmoteMDP-V.I.	(22.75, 30.0, 15.0, 10.0, 20.0, 60.0, 5.0, 20.0, 0.0)	6.009e-05	7	3.075e-05
MDPToolBox-V.I.	(22.75, 30.0, 15.0, 10.0, 20.0, 60.0, 5.0, 20.0, 0.0)	0.00075578	7	0.00078749
Gurobi	(22.7478, 30.0, 15.0, 10.0, 20.0, 60.0, 5.0, 19.997, 0.0)	0.02	0	0.023

Conclusion
Nous obtenons donc des résultats qui correspondent au résultat attendu. MDPToolBox ne dispose pas de option de minimisation, on ne peut donc pas traiter un problème de ce type tel quel mais il faut multiplier la matrice de récompense par -1.
MarmoteMDP reste le solveur le plus rapide pour ce problème. Pour répondre au problème posé de la section précédente, on peut interroger MarmoteMDP sur la politique à laquelle correspond ce résultat. On obtient: (2,4,4,5,5,5,6,7,3). Ce sont les actions à prendre pour optimiser le temps de trajet pour aller au collège.

6 Gym - Passage à l'échelle

6.1 Frozen Lake: Modèle

Bien que les exemples vus précédemment sont intéressants, ils ne sont pas de taille suffisamment importante pour pouvoir juger les performances des différentes méthodes de résolution. Ainsi nous avons choisi des problèmes modélisable sous forme de MDP, toujours simple, mais de taille plus grande. Le

problème Frozen Lake en est un exemple: Un agent doit trouver un chemin d'un état de départ à un état d'arriver en évitant des "trous". Ainsi le problème sous forme de grille de taille 4×4 comporte 16 états possible et 4 actions possible pour chaque état (gauche, droite, bas, haut). Notre modèle est constitué donc de 4 matrices de transitions 16×16 et d'une matrice de récompense 16×4. On dispose dans gym aussi d'un modèle Frozen-Lake8*8 identique de taille 8×8. On s'aperçoit que le nombre d'états croit exponnentiellement avec le nombre de cases. On aura dans ce nouveau problème des matrices de transitions de taille 64×64.

6.2 Résolution

On utilise pour ce problème le critère total à Horizon Infini. **Frozen-Lake**
On pose un nombre de pas maximal à 1000 et un $\epsilon = 0.0001$

Frozen-Lake4*4:				
Algorithme:	V(S)	T Cal	Itérations	T Mod
MarmoteMDP-V.I.	(2.468240, 2.467454, 2.466895, 2.466605, 2.468411, 0, 1.586557, 0, 2.468741, 2.46920, 2.292897, 0, 0, 2.646076, 2.823020, 0)	0.001143	288	4.571e-05
MDPToolBox-V.I.	(2.468240, 2.467453, 2.466894, 2.466604, 2.468411, 0, 1.586556, 0, 2.468740, 2.469204, 2.292897, 0, 0, 2.646076, 2.823019,0)	0.0139	288	0.00056
Gurobi	(2.458777, 2.456567, 2.455094, 2.454358, 2.459515, 0, 1.580852, 0, 2.460991, 2.46320, 2.28793, 0, 0, 2.64142, 2.820440, 0)	0.04	14	0.03

On prend maintenant le même problème mais de taille 8x8. Comme il y a 64 états, on va afficher que les 5 premiers éléments de V(S).

Frozen-Lake8*8:				
Algorithme:	V(S)	T Cal	Itérations	T Mod
MarmoteMDP-V.I.	(2.998385, 2.998456, 2.998558, 2.998669, 2.998781, 2.998888...)	0.01469	548	5.555e-05
MDPToolBox-V.I.	(2.998385, 2.998456, 2.998557, 2.998669, 2.998781...)	0.02977	548	0.00054
Gurobi	(2.965484, 2.966374, 2.967695, 2.969203, 2.970814...)	0.03	96	0.025

Conclusion
On s'aperçoit que pour des problèmes un peu plus grand les performances de MarmoteMDP et MDP-ToolBox se rapprochent, en précision et en temps de calcul. MarmoteMDP a un temps de construction du modèle toujours beaucoup plus court.

6.3 Taxi-v3: Un problème plus grand

Nous allons maintenant regarder un exemple de problème modélisable sous forme de MDP disponible sur Gym, avec une taille encore plus importante: 6 matrices de transitions de taille 500×500 (500 états et 6 actions possibles), et une matrice de récompense de taille 500×6. Comme le tableau V(S) comporte aussi 500 valeurs nous allons représenter les 4 premières dans le tableau. On utilisera comme critère le critère escompté.

6.3.1 Résolution:

On pose $\gamma = 0.5$, un nombre de pas maximal à 1000 et un $\epsilon = 0.0001$

Taxi:				
Algorithme:	V(S)	Temps Cal	Itérations	Temps Mod
MarmoteMDP-V.I.	(11.999989,-1.945324, -1.125011, -1.890650)	0.00301	20	4.34193e-05
MarmoteMDP-P.I.M	(12.0,-1.945312, -1.125000, -1.890625)	0.004428	17	4.506542e-05
MarmoteMDP-V.I.GS	(12.0,-1.945313, -1.125001, -1.890631)	2.410e-05	15	6.341934e-05
MDPToolbox-V.I.	(11.999954, -1.945358, -1.125045, -1.890724)	0.0231926	18	0.023622
MDPToolbox-P.I.	(12.0, -1.9453125, -1.125, -1.890625)	0.234582	16	0.0044982
MDPToolbox-V.I.GS	(11.999996, -1.945362, -1.125049, -1.891021	0.12650	11	0.03830
MDPToolbox-PL	(-2.000000, 9.199999, -2.000000, -1.953125	10.69466	0	0.009066
Gurobi-PL	(12.0, 0.0, 0.0, 0.0)	0.0275	44	0.035

7 Conclusion

Le domaine des processus décisionnels de Markov, avec les modèles de décision, critères d'optimalité et algorithmes d'optimisation que nous venons de présenter constitue un outil méthodologique de base en intelligence artificielle. Il est devenu incontournable pour concevoir et analyser les méthodes formelles développées aujourd'hui sur le thème de la décision séquentielle dans l'incertain.
Nous avons ainsi vu dans ce projet d'abord les bases théoriques des Processus de Décisions Markoviens, afin de pouvoir comprendre les différentes méthodes de résolutions et fonctionnements des solveurs. Ayant choisi de faire une étude comparative de deux solveurs de MDP (MarmoteMDP,MDPToolBox) et un solveur de programme linéaire(Gurobi), nous avons présenté leur différents fonctionnements ainsi que leurs avantages et inconvénients. Après avoir défini un exemple de problème de MDP par critère de résolution, et les avoir modélisés de manière adaptée à chaque solveur, nous avons pu comparer les performances et caractéristiques des solveurs: Si l'on considère l' **efficacité**, MarmoteMDP s'est montré comme le solveur le plus rapide. Il a été sur la totalité des exemples 10 à 100 fois plus rapide que MDPToolBox. Concernant la programmation linéaire, l'approche de calcul est différente et de cela non comparable. Elle est par définition moins efficace. On étudie les MDP à travers la résolution de la Programmation Linéaire à cause de son approche élégante et la liberté qu'elle offre par rapport à l'ajout de contraintes et l'analyse des résultats. En effet, un MDP de 10 000 états serait considéré comme un grand problème difficile à résoudre linéairement alors que cela n'est toujours pas le cas pour les solveurs d'MDP. MarmoteMDP offre aussi des résultats plus **consistant** en temps et précision des calculs (peu de différences entre les méthodes choisies). En comparant la **facilité d'utilisation**, MarmoteMDP et MDPToolBox restent les deux assez simples à utiliser. MDPToolBox serait plus adapté si l'on souhaite résoudre rapidement un problème de maximisation (puisque on ne peut pas faire de minimisation) de taille modérée, grâce à son interface simpliste et au nombre réduit de méthodes utilisées. MarmoteMDP serait le solveur à choisir pour une approche plus analytique car il offre beaucoup plus de liberté dans la modélisation du problème et offre des résultats plus détaillés.

8 Ouverture

Dans cette section nous allons brièvement présenter les idées de poursuite de recherche que nous n'avions pas eu le temps de réaliser:
Faire *évoluer la taille des modèles*: En effet, les exemples que nous avons traités sont petits et ne nous ont pas permis de bien comparer les performances des 3 méthodes de résolution de MDP. En suite, on pourrait comparer ces modèles avec d'autres ayant des matrices très creuses. On suppose que cela pourrait impacter la vitesse de construction des modèles par les solveurs:

- Axe 1: Des modèles avec beaucoup d'états de sortie (matrices pleines) et peu d'actions (peu de matrices).

- Axe 2: Des modèles avec peu de sorties (matrices creuses) et peu d'actions (peu de matrices).
- Axe 3: Des modèles avec beaucoup de sorties (matrices pleines) et beaucoup d'actions (beaucoup de matrices).
- Axe 4: Des modèles avec peu de sorties (matrice creuses) et beaucoup d'actions (beaucoup de matrices).

Nous pourrons prendre des modèles issus de Gym, des modèles utilisés pour benchmarker des algorithmes de RL, ou des modèles issus de Prism.

9 Bibliographie

[1] F.Garcia, Chapitre 1 de « Processus Decisionnels de Markov en Intelligence Articielle ». Hermes, 2008.

[2] W. B. Powell, Chapter 3 of Approximate Dynamic Programming: Solving the Curses of Dimensionality. John Wiley Sons, 2007.

[3] M. L. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley Sons, 2014.

[4] A. Malek, Y. Abbasi-Yadkori, and P. Bartlett, "Linear Programming for Large-Scale Markov Decision Problems," in International Conference on Machine Learning, Jun. 2014, pp. 496–504, Accessed: Apr. 01, 2021. [Online]. Available: http://proceedings.mlr.press/v32/malek14.html.

[5] S. Brooks, A. Gelman, G. Jones, and X.-L. Meng, Handbook of Markov Chain Monte Carlo. CRC Press, 2011.

[6] J. Si, A. G. Barto, W. B. Powell, et D. Wunsch, Handbook of Learning and Approximate Dynamic Programming. John Wiley Sons, 2004.

[7] E. L. Porteus, Foundations of Stochastic Inventory Theory. Stanford University Press, 2002.

[8] R. J. Boucherie et N. M. van Dijk, Markov Decision Processes in Practice. Springer, 2017.

[9] D. Wingate, « Prioritization Methods for Accelerating MDP Solvers », Journal of Machine Learning Research 6, p. 31, 2005.

[10] D. Silver et al., « Mastering the game of Go without human knowledge », Nature, vol. 550, no 7676, p. 354-359, oct. 2017, doi: 10.1038/nature24270.

[11] V. Mnih et al., « Human-level control through deep reinforcement learning », Nature, vol. 518, no 7540, p. 529-533, févr. 2015, doi: 10.1038/nature14236.

[12] O. Sigaud et O. Buffet, Markov Decision Processes in Artificial Intelligence. John Wiley Sons, 2013.

[13] G. B. Dantzig and M. N. Thapa, Linear Programming 1: Introduction. Springer Science Business Media, 2006.