Etude comparative de solveurs pour les Processus de Décisions Markovien

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

petarcalic99@gmail.com

19/05/21

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 trois solveurs que nous allons étudier. Nous allons ensuite préparer un benchmark qui doit être appliqué à tous les solveurs. La dernière phase est celle où on doit faire tourner les solveurs sur nos exemples et analyser les résultats obtenus.

Mots-clés: MDP; Solveurs; 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 intéressant. La partie de construction de benchmark pour une analyse complète ne sera donc pas présente dans le rapport.

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 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) 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 aussi pas les mêmes.

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 demie 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)
  - $\Re : S \times \mathrm{A} \to \Re$ (récompense déterministe rattachée à l'action en ignorant son résultat)
  - $\Re : \mathrm{S} \to \Re$ (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 l'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 r, a été obtenue par rapport aux états s et actions a effectuées, l'MDP peut être représenté ainsi :

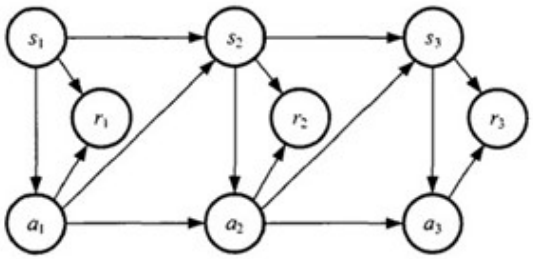

Figure 2: MDP et récompenses.

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.

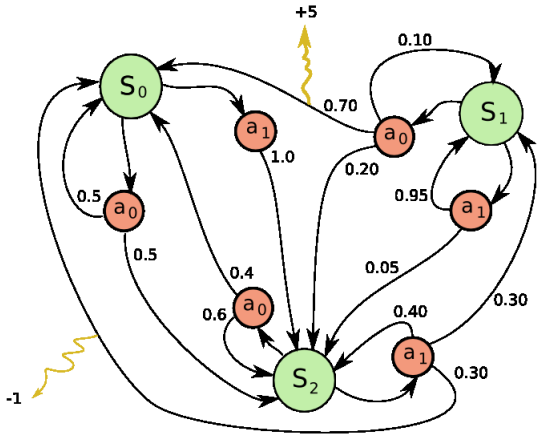

Figure 3: Exemple d'un MDP.

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 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 on devrait prendre en compte une nouvelle variable de temps. Lorsque ces fonctions de varient pas, on parle de *processus stationnaires.*
Les processus décisionnels ne 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 (noté π).* Appelée aussi stratégie, c'est la procédure suivie par l'agent pour choisir à chaque instant l'action à exécuter. On peut faire deux distinctions ici. Une politique peut déterminer précisément l'action à effectuer selon l'état où 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, 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 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 »).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 choisi un critère et une politique, nous pouvons déterminer 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 entrée par état. Nous pouvons aussi définir une autre fonction de valuation dite états-actions : $Q^{\pi}(s, a)$:

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

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

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

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

Ceci correspond a l'opérateur pour le critère escompté, il faut légèrement adapter la formule selon le critère. Avec V la valeur d'être à cet état r la récompenses de prendre une action et p 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 qu'on a 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 optimales 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 valeur peut donc être approché par un algorithme itératif.
En effet si on choisit d'utiliser le critère discompté, on peut écrire la fonction de valeur ainsi:

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

2.7 Algorithmes

Les MDP peuvent être résolus de différentes manières. Les algorithmes principaux 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.

Ainsi on appelle 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 initial, on convergera vers la Value Fonction optimale:

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

Ainsi on doit se poser un ϵ très petit:

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

Pour obtenir $V^* = V_n$ L'algorithme en détail peut être vu dans le chapitre 1 de F.Garcia. Ensuite on en déduit la politique ainsi:

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

2.7.2 Policy Iteration:

L'algorithme d'itération sur la politique est similaire à l'algorithme précédent mais il est en deux phases: 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}:
$\pi_0 = 0$
$V_0 = 0$

$$\pi_0(s) \leftarrow \arg\max_{a \in A}[r(s, a) + \gamma \sum_{s' \in S}^{\infty} [\ p(s'|a, s)V_0^\pi(s')]]$$

Évaluation de la politique:

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

Amélioration de la politique:

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

Le fait de faire une évaluation à chaque itération étant très lent, un algorithme dit 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 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 (ici c'est un critère escompté):

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

Sous Contraintes:

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

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 $|S|, |A|, b$ 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

Nous détaillons ici pour chacun des critères, les types d'algorithme que l'on peu utiliser.

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 utilisé. 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 télechargable 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. L'intégralité des méthodes pour résoudre le problème sont implémentés et prêtes à appliquer. Le logiciel retourne aussi beaucoup d'informations supplémentaires liée 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 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.

Le module fonctionne sous python et nécessite une installation préalable des modules Scipy et Numpy. Ensuite il y a deux manières de récuperer 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 de la bibliothèque d'algorithmes la plus complète. Une documentation très riche en définition et exemple d'utilisation 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
Un de ses inconvénients est qu'il est développé uniquement pour le langage Pyhton.

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 souhaitant s'entraîner ou faire des recherches. 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 implémentées. Python est le langage de notre choix 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.
Difficile de modéliser un MDP sous forme de programme linéaire qui sera résolu par Gurobi.Aussi son utilisation pour la résolution de MDP nécessite d'être familier avec le solveur Gurobi puisque sont fonctionnement est assez particulier.

3.4 Comparaison des solveurs

Chacun des trois logiciels a ses avantages et ses inconvénients et sera utilisé par rapport au problème et l'angle de résolution que l'on souhaite. ToolBoxMDP dispose de l'interface la plus simple à utiliser et d'une bibliothèque de fonctions la plus complète. Or ce module offre peu de liberté dans l'organisation et la manipulation des calculs. L'utilisateur n'a par exemple pas de moyen d'agir sur le solveur de programme linéaire que utilise ToolBox et fonctionne plutôt sous le principe de boite noire. Aussi les résultats retournés sont beaucoup moins détaillés que chez MarmoteMDP,qui est sans doute le solveurs le plus adapté pour une étude plus détaillé sur les MDP puisque le programme retourne avec les résultats du problème une multitude de données pertinentes pour une analyse numérique. L'interface d'utilisation de MarmoteMDP demande plus de préparations que ToolBoxMDP mais est plus transparente.
Pour une résolution par programmation linéaire l'option plus rapide est sans doute d'utiliser ToolBoxMDP qui a des méthodes prédéfinies, mais pour une étude plus sérieuse il faudrait utiliser Gurobi qui est un solveur de programme linéaire uniquement et demande que les programmes soient modélisés manuellement.

3.5 Plateforme de comparaison

Afin de pouvoir 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 Virtuel 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 d'utiliser et de manipuler des Machines Virtuelle puisque plusieurs problèmes étaient survenus tel que une mémoire et un espace disque insuffisant.

4 Quelques problèmes de Décision

4.1 Critère escompté

4.1.1 Modélisation - MarmoteJouet10

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 récompenses: On représente chacune des matrices associées à une action. Il s'agit des matrices P_1 associée à la décision 1 et P_2 associée à la décision 2.

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

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 si-dessous en annexe. Néanmoins, évoquons quelques points importants:

- MarmoteMDP: La résolution d'un MDP peut être divisée en deux étapes principale dans MarmoteMDP. On commence par construire le modèle. On créer les matrices de transitions et de coût. On initialise aussi toutes les constantes nécessaire. A l'aide de la méthode MDP1 = discountedMDP(critere, stateSpace, actionSpace, trans, Reward,beta) on a définit 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 qui a eu lieu 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'exécutions.
- MDPToolbox: Ce solveur a une structure très similaire à MarmoteMDP. Il y a deux étapes principales: Celle de la construction de l'MDP, puis celle de la résolution de ce dernier. Les différences son plutôt formels (types et arguments appelés par les méthodes) et la méthode de résolution (vi.run()) ne retourne rien. L'objet "vi" est surchargé et il faut appeler tout élément dont on a besoin.
- Gurobi: Gurobi demande d'avoir des pré-requis en programmation linéaire. Il faut savoir modéliser le problème et ensuite utiliser Gurobi. Mais la résolution d'un problème reste assez élégant une fois ces points acquis. Tout d'abord il faut définir toute les matrices et paramètres du problèmes identiquement aux cas précédents. 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 de la contrainte sur les v(s): borne inférieur de l'équation de Bellman.

  Voici le modèle:

$$z = \min_V \ (v_1 + v_2)$$

$$s.c. \quad v_i \geq R[i,j] + \gamma \cdot P[j][i,j] \cdot v_i \quad \forall i,j \in [1,2]$$
$$v_1, v_2 \in \Re$$

4.2 Critère moyen

MarmoteJouet20
Ce modèle correspond à l'exemple 8.5.3 du livre de Puterman. Même si il reste simple, c'est un exemple très pratique et important puisque il a été calculé formellement. Ayant obtenu des résultats différents de fonctions de évaluations pour le premier exemple nous avons choisi cet exemple pour vérifier que les algorithme 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 un 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 possible 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 d'une part (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

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

- MarmoteMDP:
- MDPToolbox:
- Gurobi:

4.3 Critère total

MarmoteJouet40
C'est 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 $M = 3$, l'espace d'action est positif et aussi de taille 3. La matrice de revenu est :

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

Et voici les matrices de transitions:

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

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

4.3.1 Code

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

- MarmoteMDP: On manipule ici des MDP à horizon fini et critère total, *finiteHorizonMDP*. La seule méthode qui existe pour ce genre de modèle est Value Iteration. Les autres sont là par souci de compatibilité et renvoient des politiques non pertinentes. On peut noter qu'il y a qu'une seule matrice utilisée et que donc le nettoyage doit faire appel aux références qui sont dans le vecteur.
- MDPToolbox:
- Gurobi:

5 Résultats numériques

5.1 Critère escompté

MarmoteJouet10:
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 $\epsilon = 0.0001$

MarmoteJouet10:				
Algorithme:	V(S)	Temps(s)	Itérations	x
MarmoteMDP-V.I.	(8.428541,6.999970)	4.053e-05	18	
MarmoteMDP-P.I.M	(8.428555 ,6.999983)	2.336e-05	3	
MarmoteMDP-V.I.GS	(8.428556 ,6.999991)	2.813e-05	15	
MDPToolbox-V.I.	(8.18, 6.75)	0.00023	5	
MDPToolbox-P.I.	(8.43, 7.0)	0.0036	1	
MDPToolbox-V.I.GS	(8.42, 6.99)	0.00033	7	
MDPToolbox-PL	(8.4285, 6.9999)	0.028	0	
Gurobi-PL	(6.429,3.529)	0.04	0	

Conclusion: Même si il s'agit d'un exemple très petit, et qu'on ne peu pas tirer de conclusion définitive, on a bien certaines indication sur le comportement des solveurs. MarmoteMDP semble être le plus précis et s'approche le plus de la valeur théorique attendu. C'est aussi le plus consistent. Les trois algorithme retourne un peu près la même valeur. MDPToolbox a de bons résultat pour Policy Iteration et Gauss Seidel Value Iteration, mais on peu considérer que la précision de Value Iteration 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.

5.2 Critère moyen

MarmoteJouet20
Résultats Attendus: On doit obtenir 2.5 comme coût moyen. Ce résultats est calculé formellement dans le livre de Puterman.

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

MarmoteJouet20:				
Algorithme:	V(S) *moyen*	Temps(s)	Itérations	x
MarmoteMDP-V.I.	2.378235	0.000711	1000	
MarmoteMDP-P.I.M	2.260885	0.000821	1000	
Gurobi-PL				

On s'aperçoit que les calcules sont lent et pas précis. Il y a du surapprentissage.
On diminue donc $\epsilon = 0.01$:

MarmoteJouet20:				
Algorithme:	V(S) *moyen*	Temps(s)	Itérations	x
MarmoteMDP-V.I.	2.498924	2.909e-05	1000	
MarmoteMDP-P.I.M	2.498121	3.361e-05	1000	
Gurobi-PL				

5.3 Critère Total

MarmoteJouet40
Résultats Attendus:
On pose un nombre de pas maximal à 1000 et un $\epsilon = 0.0001$

MarmoteJouet40:				
Algorithme:	V(S)	Temps(s)	Itérations	x
MarmoteMDP-V.I.	(0,5,6,5)	3.790e-05		
MarmoteMDP-P.I.M				
Gurobi-PL				

6 Conclusion

7 Bibliographie