Résolution de jeux

Projet de Master 1

Encadrant : Emmanuel HYON

Réalisé par :

Yoones MIRHOSSEINI, Marc VINCENT

Master 1 Androïde

Sorbonne Université

Janvier – Juin 2018

SOMMAIRE

1 – Descriptif général

Le sujet de ce projet porte sur les jeux stochastiques à somme nulle à deux joueurs. L'objectif est dans un premier temps de concevoir une modélisation informatique de ces jeux et dans un second temps d'implémenter et comparer des algorithmes permettant de résoudre ces jeux. Les algorithmes testés seront des types suivants : par apprentissage par renforcement, par méthode de Newton et par programmation linéaire. Le but de ces algorithmes est de trouver l'équilibre de Nash en stratégie mixte s'il existe, afin d'en déduire la ou les stratégie(s) optimale(s) pour chaque joueur. Trois jeux serviront à tester nos implémentations : pierre-papier-ciseaux, une version simplifiée du jeu de foot et un jeu d'attaque-défense dans un graphe.

2 – Contexte

Dans ce projet, nous faisons usage de la notion de processus de décision markovien (MDP). Un MDP est la formalisation d'un « processus de décision séquentielle dans l'incertain » ; il s'agit formellement d'un « processus stochastique contrôlé satisfaisant la propriété de Markov », défini par un espace d'états, un espace d'actions, un axe temporel, des probabilités de transition entre états et une fonction de récompense. Lorsque l'on se trouve dans un état s et que l'on effectue une action a, on reçoit une récompense $r(s, a)$ et on passe à un autre état selon une distribution de probabilités $p(s' \mid s, a)$. La solution d'un MDP est une stratégie permettant de maximiser les gains obtenus. Le principe d'une stratégie, également appelée « politique » ou « règle de décision », est de fournir l'action que l'on réalisera à chaque étape ; dans le cas d'un MDP, il s'agit toujours d'une stratégie mixte (c'est-à-dire probabiliste) et non pure (déterministe).

Ce projet s'inscrit également dans le large contexte de la théorie des jeux, qui est « un formalisme qui vise à étudier les interactions entre individus » et qui couvre un très large champ d'applications. Dans un jeu, chaque joueur doit choisir la meilleure action possible à réaliser, ce qui se mesure généralement par une fonction associant un gain à chaque action de tous les joueurs : cette action se décide en fonction de la stratégie du joueur. Le but est généralement de trouver un équilibre de Nash, c'est-à-dire une action conjointe dont aucun joueur n'a intérêt à dévier : si un joueur i change son action et que les autres conservent celle qui est la leur dans l'équilibre, le gain du joueur i sera moindre. Il existe de nombreux types de jeux différents ; ceux qui nous intéressent ici sont du type suivant :

- non-coopératif : pas de possibilité de passer des accords entre joueurs pour trouver un compromis de gain ;
- à information complète : chacun connaît les stratégies possibles et gains des autres ;
- à somme nulle : la somme des gains des joueurs est nulle.

De plus, les jeux que nous étudierons sont des jeux stochastiques. Les jeux stochastiques sont un modèle réunissant les jeux dynamiques répétés et les MDP. Un jeu dynamique répété est un jeu étendu dans le temps, avec une action à réaliser à chaque tour, sachant que les joueurs choisissent simultanément leur action. Ainsi, dans un jeu stochastique, chaque action conjointe mène à un nouvel état où se joue l'équivalent d'un jeu statique avec des gains particuliers à chaque état.

La convergence des jeux stochastiques à somme nulle vers un équilibre de Nash pose problème et de nombreux algorithmes ont été proposés pour pallier cela. Nous détaillons ci-dessous les types d'algorithmes que nous étudierons.

2.1 – Algorithme de Shapley avec programmation linéaire

Dans un premier temps nous allons utiliser et implémenter l'algorithme de Shapley [SHA 4] pour résoudre les jeux stochastiques à deux joueurs à somme nulle.

Cet algorithme trouve un équilibre de Nash, puis à partir de cet équilibre de Nash on est capable de calculer les stratégies optimales pour chaque joueur.

Cet algorithme fonctionne selon le principe de l'itération sur les valeurs : à chaque itération, pour chaque état de jeu possible, il faut calculer la valeur du jeu, et ce jusqu'à convergence pour tous les états, pour finalement calculer des stratégies optimales à partir des valeurs trouvées. L'algorithme originel utilise l'énumération de Snow et Shapley qui est très coûteuse en temps. Pour résoudre ce problème nous allons utiliser la programmation linéaire pour calculer la valeur du jeu et ainsi que les stratégies optimales.

La programmation linéaire est une technique mathématique d'optimisation de fonction objectif linéaire sous des contraintes ayant la forme d'inéquations (ou équations) linéaires. [BAU 5]

Pour calculer la valeur de jeu à chaque étape (état du jeu) nous allons résoudre la fonction ci-dessous après transformation en forme linéaire.

$$v(s) = max_{\pi^1 \epsilon \Pi^1} min_{\pi^2 \epsilon \Pi^2} \sum_{a^1 \epsilon A^1} R(s, v, a^1, a^2) \pi^1(a^1)$$

où $\pi^1(\pi^2)$ est la stratégie du joueur 1 (respectivement 2) et $\Pi^1(\Pi^2)$ l'ensemble des stratégies du joueur 1 (resp. 2), $a^1(a^2)$ l'action effectuée par le joueur 1 (resp. 2), A^1 l'ensemble des actions disponibles pour le joueur 1, $R(s, v, a^1, a^2)$ la récompense obtenue et finalement $\pi^1(a^1)$ la probabilité de choisir l'action a^1 dans la stratégie π^1 . *v(s)* est la valeur du jeu pour l'état s.

2.2 – Apprentissage par renforcement

Les méthodes d'apprentissage par renforcement permettent de traiter les situations dans lesquelles les fonctions de transition et de récompense d'un MDP ne sont pas connues a priori [PDMIA 3]. La plupart de ces méthodes s'inspirent des principes de l'apprentissage humain ou animal : renforcer la tendance à exécuter une action dont les conséquences sont jugées positives, prendre en compte la durée qui sépare la récompense de l'action ou la fréquence à laquelle cette action a été testée. [PDMIA 3] L'apprentissage par renforcement est donc une forme d'apprentissage non supervisé reposant sur une évaluation, qui implique un fonctionnement par essais et erreurs. Ainsi, *« l'apprentissage par renforcement est basé sur une interaction itérée du système apprenant avec l'environnement, à partir de laquelle une politique est progressivement améliorée. Cependant, en pratique, la plupart des algorithmes d'apprentissage par renforcement ne travaillent pas directement sur la politique, mais passent par l'approximation itérative d'une fonction de valeur ».* [PDMIA 3]

L'algorithme d'apprentissage par renforcement que nous allons implémenter, proposé par Littman [LIT 1] et connu sous le nom de « Minimax Q-learning », est une extension de l'algorithme de Q-learning standard. Le Q-learning est une méthode d'apprentissage par renforcement qui consiste à apprendre la valeur des actions selon les états, ce qui permet de calculer les utilités et la politique optimale dynamiquement.

En Q-learning, on associe à chaque couple état-action d'un agent une Q-valeur Q(s, a) qui correspond à « *la récompense espérée obtenue en effectuant l'action a dans l'état s et en suivant une politique optimale à partir de l'état suivant.* » [GIE, CHA, 2] Ces valeurs sont évaluées dynamiquement par l'expérience de l'agent dans l'environnement. Le Q-learning est un algorithme d'apprentissage mono-agent, qui peut être adapté pour un environnement multi-agent mais sans prendre en compte explicitement la présence des autres agents : en effet, dans ce cas, l'autre agent est considéré comme une partie de l'environnement.

Le principal problème que pose l'application directe du Q-learning est le suivant : dans le cas où l'adversaire a une stratégie complète et complexe, il est possible que l'algorithme ne converge pas, ce qui nous empêche de trouver la stratégie optimale. L'algorithme Minimax Q-learning essaye de résoudre ce problème et de prendre en compte de manière plus explicite la présence de l'autre agent en utilisant la technique du minimax de la théorie des jeux.

2.3 – Jeux-jouets

Les jeux-jouets sur lesquels nous testerons nos approches sont des exemples classiques des jeux stochastiques :

- pierre-papier-ciseaux est un exemple de jeu stochastique à un état, avec 3 actions disponible, soit pratiquement le modèle le plus simple possible. C'est ainsi un cas particulier où il n'y a pas de part d'aléatoire dans l'évolution du jeu, c'est donc l'équivalent d'un jeu dynamique répété ;

- le jeu de foot simplifié implique deux joueurs et un ballon sur un terrain composé de 4x5 cases. Chaque agent contrôle un joueur et a au maximum 5 actions possibles (déplacement vers une case adjacente ou immobilité) : les 2 actions sont choisies

simultanément mais l'ordre dans lequel elle s'exécute est aléatoire. Le ballon se déplace en fonction des actions des joueurs, avec une issue aléatoire à certaines actions conjointes (quand les joueurs essaient de se déplacer vers la même case par exemple). Le but est évidemment d'aller porter le ballon à une extrémité du terrain ;

- l'attaque-défense dans un réseau électrique : un réseau électrique peut être représenté par un graphe avec des nœuds qui créent de l'électricité, des nœuds qui en consomment et des arcs qui assurent le transport de l'électricité. Le but d'un attaquant est de minimiser la quantité d'électricité transportée, celui d'un défenseur est de la maximiser. L'action à chaque tour d'un attaquant est d'endommager un nombre limité de liens entre les nœuds du réseau, celle d'un défenseur est de le contrer en réparant ou en renforçant un nombre limité de liens. Pour chaque paire d'actions, la transition vers le nouvel état s'effectue selon une distribution de probabilité.

3 – Tâches à réaliser

Les tâches nécessaires à la réalisation du projet incluent :

• une bibliographie portant sur les bases théoriques des jeux stochastiques et sur les algorithmes de résolution proposés dans la littérature ;

• une phase de conception et de modélisation de l'information, afin permettre une implémentation générique des jeux stochastiques à deux joueurs à somme nulle ;

• une définition et instanciation des jeux de tests que nous utiliserons : pierre-papier-ciseaux, foot simplifié, attaque-défense (optionnel) ;

• une étude des *use cases* de notre logiciel, c'est-à-dire des exemples d'utilisation pour aider à prendre en main le logiciel ;

• une implémentation des algorithmes de résolution :

- par l'algorithme dit de Shapley, avec utilisation de la programmation linéaire pour la résolution du jeu statique à chaque étape ;
- par l'algorithme Minimax-Q-learning, basé sur l'apprentissage par renforcement ;
- par l'adaptation d'une méthode de Newton (optionnel) ;

• la rédaction d'une documentation, du même genre que Java Doc, permettant de faciliter l'utilisation du logiciel ainsi que les modifications ultérieures par d'autres personnes ;

• une phase de tests et de comparaison sur les critères suivants : vitesse de convergence, qualité de la solution trouvée et surtout vérification de la convergence vers une solution ; en effet, cette convergence n'est pas garantie par tous les algorithmes traitant les MDP multi-agents. On fera notamment jouer deux agents avec deux différentes stratégies (trouvées par deux différentes méthodes) pour comparer le taux de victoire de chaque agent ;

• optionnellement, nous ferons la comparaison avec une implémentation existante en C++.

4 – Aspects techniques

Nous avons décidé que le langage utilisé dans le projet serait Python parce qu'il nous est familier, qu'il permet la programmation objet, qu'il est portable et largement répandu et qu'il dispose de nombreuses bibliothèques qui faciliteront l'implémentation.

Nous nous livrerons également à des analyses de la complexité et des performances empiriques des algorithmes implémentés.

Afin de pouvoir mener une comparaison avec l'implémentation existante, nous utiliserons SWIG pour transférer celle-ci de C++ à Python.

Pour résoudre les programmes linéaires, nous utiliserons Gurobi, un solveur d'optimisation pour la programmation mathématique comme la programmation linéaire, quadratique et etc. Gurobi possède plusieurs implémentations en différents langages de programmation. Nous utiliserons l'interface Python de Gurobi.

5 – Proposition d'organisation

La bibliographie et le rapport seront élaborés tout au long du projet. Nous nous fixons par ailleurs les dates suivantes pour mener à bien les différentes étapes du projet :

• début mars : complétion du cahier des charges ;

• fin mars : modélisation et algorithme de Shapley traités ;

• fin avril : apprentissage par renforcement traité (méthode de Newton optionnelle) ;

• fin mai : tests et rapport.

Références

[LIT 1] LITTMAN M., « Markov Games as a Framework for Multi-Agent Reinforcement Learning », Proceedings of the Eleventh International Conference on Machine Learning (ICML'94), 1994.

[GIE, CHA 2] O. Gies , B. Chaib-draa « Apprentissage de la coordination multiagent : Q-learning par jeu adaptatif »

[PDMIA 3] Groupe PDMIA « Processus Décisionnels de Markov en Intelligence Artifcielle » , 2008

[SHA 4] SHAPLEY L.S.., « Stochastic Games », Proceedings of the National Academy of Sciences of the United States of America (PNAS), vol. 39, p. 1095–1100, 1953.

[BAU 5] William J. Baumol, Economic theory and operations analysis, 4ème edition, Harper & Brothers, New York,