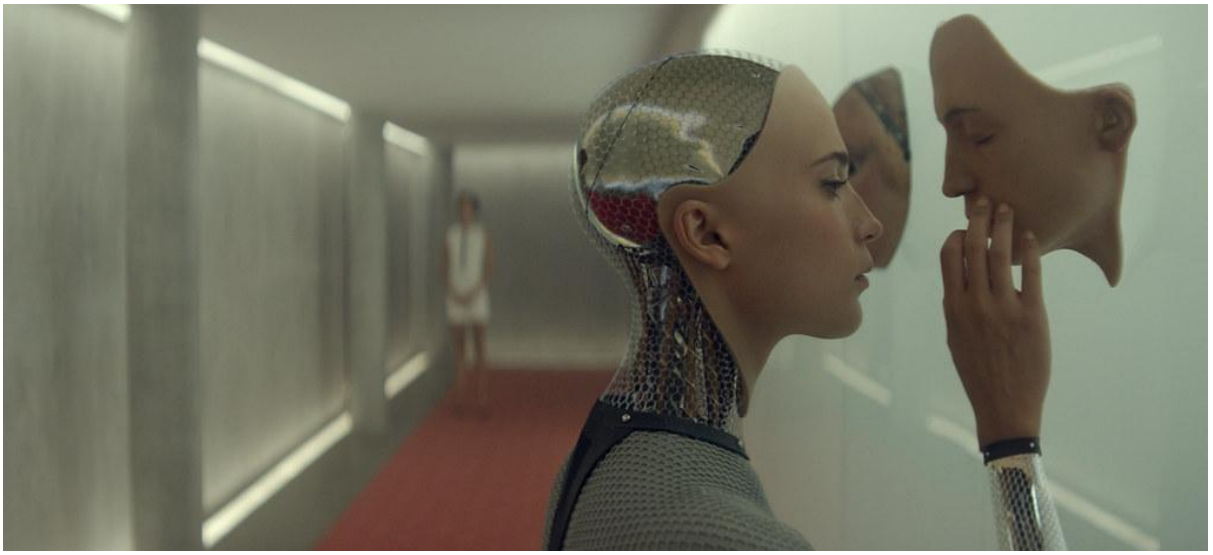
# Intelligence Artificielle



Encadrant : Marc-Michel CORSINI

*L3 MIASHS - 2019/2020*

*Marine TROADEC*
*Valentin STRAHM*
*Thomas SUBRENAT*

université de BORDEAUX

# ***Sommaire :***

# Introduction :

Pendant ce semestre il a été nécessaire de coder une modélisation du dilemme du prisonnier ainsi que des stratégies possibles pour lesquelles un individu aurait pu opter.
Le dilemme du prisonnier peut se résumer simplement, il s'agit de deux individus ayant deux choix disponibles (duper ou coopérer) et ne sachant pas ce que l'autre va jouer. Une fois que chacun a fait son choix, ils se voient attribuer une récompense en fonction de leur choix ainsi que de celui de leur adversaire.
Nous en venons donc à notre problématique.

***Pouvons nous trouver et si oui quelle serait la stratégie qui maximiserait le score sur un modèle de mémoire 2 ?***

# 1 - Les enjeux et critères d'efficacité

## 1) Objet d'étude

Le dilemme du prisonnier est un jeu très fréquenté dans la théorie des jeux. Celui ci, est utilisé dans de nombreuses situations de la vie courante. Par exemple, il peut être utilisé pour des questions d'économie, de mathématiques et même de biologie. Le dilemme du prisonnier, créé en 1950, a pour principe de faire affronter une personne face à une autre avec comme but une récompense.

Celle-ci peut être nulle, partagée, gagnée entièrement par le joueur 1 ou gagnée entièrement par le joueur 2.

En effet, la récompense étant très souvent de l'argent, les joueurs, pour maximiser leurs gains respectifs doivent coopérer entre eux sans communiquer. Pour éviter toute représaille de la part d'un des deux joueurs il suffit d'être honnête et non manipulateur.

Cependant, il peut y avoir une différence lorsque le jeu se joue une seule fois, n fois ou bien une infinité de fois car si nous jouons n fois des situations peuvent se ressembler et même être identique, ce qui nous laissera supposer ce que jouera l'adversaire dans la partie suivante, et donc les récompenses à la clef. Autrement dit, le joueur peut observer le comportement adverse afin d'adapter sa stratégie et ainsi obtenir le plus de gains.

Le but étant de maximiser le gain il se peut qu'en arrivant à la dernière partie les joueurs deviennent moins coopératif et changent leur habitudes pour leur intérêt général. Cela reflète une vraie étude des comportements chez les joueurs :

- Pourquoi joue-t-il plutôt ça que ça,
- Comment a t-il pensé à jouer ça,
- Et si il avait joué autrement ?
- Pourquoi ai-je arrêté de coopérer ?

Dans notre projet, le dilemme du prisonnier, reproduit algorithmiquement, contient 2 choix principaux qui s'offrent aux joueurs : coopérer ou duper. Les choix des joueurs sont

déterminant dans la récompense qu'ils vont obtenir . Celles-ci sont T>R>S>P et 2R>T+S. Par exemple une récompense R sera supérieure à S.
De ces choix et ces récompenses, un tableau des gains peut représenter les différentes solutions.
Si nous nous mettons à la place du joueur 1, alors si il joue Coopérer ( C ) et que le joueur 2 joue Duper ( D ) je gagne S et l'autre joueur T. Si je joue Duper ( D ) et que le joueur 2 joue C, alors je gagne T et l'autre joueur S.
En effet si les deux joueurs jouent C, ils gagnent tous les deux R et s'ils jouent tous les deux D, ils gagnent P.

Voici le tableau des différentes récompenses :

| (J1,J2) | Coopérer | Duper |
|---|---|---|
| Coopérer | (R,R) | (S,T) |
| Duper | (T,S) | (P,P) |

En effet, le jeu pouvant être répété, le programme nécessite une mémoire, qui bien évidemment sera vide au départ. La mémoire sera de taille 1 si elle reçoit une récompense, 2 si elle retient 2 récompenses et ainsi de suite. On peut alors définir ces combinaisons de récompenses sous forme de nombre binaire "000,001,111…". Dans le cas de notre programme la mémoire est déterminée au préalable, dans le cas de notre problématique nous nous intéresserons aux mémoires de taille 1 et 2.

Avec toutes ces informations, nous pouvons en déduire plusieurs choses en suivant un raisonnement tout à fait logique pendant le jeu. En effet, nous pouvons faire des déductions de ce qu'a joué l'adversaire car en connaissant la récompense, et en ayant choisi ma propre action (coopérer ou duper) les joueurs sont capables de savoir le choix de l'autre joueur.

Prenons un exemple qui pourrait reprendre un cas succinct pendant le déroulement de notre programme si je suis le joueur 1 et que j'ai joué "coopérer":

- la récompense retenue est "R", alors par déduction le joueur 2 à prit l'option "coopérer"

- la récompense retenue est “S”, alors par déduction le joueur 2 à prit l’option “duper”

En effet, dans ces différents cas de mise en pratique, nous sommes conscient de notre choix personnel.

Avec ce système de récursivité les joueurs peuvent même anticiper l’action de l’autre joueur en fonction de ce qu’il a joué au tour d’avant en prenant en compte qu’il reproduit les mêmes choix dans le temps de manière récursive. Alors en prenant en compte les différentes fréquences de jeu, plus particulièrement des récompenses nous pouvons envisager à l’avance ce que l’autre jouera. De plus, un joueur peut avoir un système de règles prédéfinies et ainsi suivre toujours le même schéma.

La stratégie fonctionnant ainsi est Automaton. Si le joueur a une mémoire de taille 1 le système de règles et de taille 5, par contre si sa mémoire est de taille 2 alors le système de règles est de taille 21.

Notre but est donc de trouver le meilleur système de règles afin de maximiser les gains contre un maximum de stratégies.

Nous l’étudierons sur deux modèles différents, un où il n’est pas autorisé d’abandonner et un où il est possible.

## 2) Modus operandi

Il est nécessaire pour répondre à notre problématique de définir des critères d’efficacité. Ce sont des informations qui nous permettront de savoir s’il existe une stratégie automaton qui serait efficace contre les autres stratégies. Les récompenses sont un score attribué au joueur qui suit ces règles : T>R>P>S et 2XR>T+S. Ce qui se traduit par le fait que si les deux joueurs coopèrent la somme de leur gain sera plus importante que la somme des gains de joueurs ayant trahi pour l’un et coopérer pour l’autre. Pour le deuxième modèle les récompenses suivent cet ordre T>R>A>P>S.

De ce fait, dans notre cas, une stratégie efficace est une stratégie ayant le meilleur score possible). Pour arriver à trouver une stratégie Automaton qui maximiserait le score contre toute les stratégies, nous avons utilisé un algorithme génétique qui compare les scores

des différentes stratégie automaton lors de l'affrontement avec d'autres stratégies et essaie de trouver la meilleure au fil des parties.

Le score est traduit par une variable nommée adequation qui le ramène à un intervalle [0;1]. C'est cette variable qui nous intéressera pour savoir si la stratégie a été efficace

# 2 - L'explication du code

## 1) L'application des critères

Nous avons travaillé avec des modèles d'algorithmes génétiques. Ce type d'algorithme prend en entrée une population et effectue des mutations à chaque changements de générations selons des critères imposés que nous détaillerons plus tard. L'évaluation se produit à chaque génération grâce à une fonction de fitness. La fonction de fitness est la fonction permettant de mesurer et sortir des statistiques de notre population. C'est elle qui va permettre d'évaluer l'efficacité de la stratégie grâce à la variable "adequation". Ce genre d'algorithme sont très puissants dans le domaine de la recherche de donnée, ce qui est intéressant dans notre cas afin de pouvoir retrouver les meilleurs patterns d'actions possibles.

Nous nous sommes grandement aidé des paramètres de l'étape 0 du jalon 4 pour construire notre algorithme génétique. Nous avons privilégié une fonction run_wheel_truncate à la place de run_fraction_truncate pour insérer plus d'aléatoire dans l'algorithme et ainsi explorer plus largement les possibilités. Le nb_target est un entier qui au nombre de candidats parmi lesquels on cherchera le génotype le plus proche de celui de son remplaçant. Nous l'avons laissé à sa valeur par défaut qui est de 5 car l'adéquation en variait pas significativement alors que qu'un nombre plus grand alourdi l'algorithme et le rend plus lent.

Nous avons défini le paramètre factor égale à 0,8 cela permet de conserver 80% de la génération t lors du passage à t+1. L'algorithme est non élitiste pour garder un minimum de variation

•

## 2) Les résultats

a) Afin de voir l'efficacité de notre automaton, nous avons essayé de voir si l'algorithme génétique trouve une chaîne si on combat toutes les stratégies pour "multi-eval".

Nous essayons d'affronter automaton contre toutes les autres:
[Automaton] *contre* [Gentle(), Bad(), Fool(), GentleSulky(), BadSulky(), FoolSulky(), Tit4Tat(), WinStayLooseShift(),Pavlov()]

Avec un gène de longueur 21, l'adéquation est très faible (0.0803) en abandonnant
Avec un gène de longueur 21, l'adéquation est très faible (0.0816) sans abandonner

Ces valeurs, se rapprochant fortement de 0 et non de 1 montre un résultat de mauvaise adéquation.

Avec un gène de longueur 5, l'adéquation est très faible (0.0817) en abandonnant
Avec un gène de longueur 5, l'adéquation est très faible (0.0800) sans abandonner

Ces valeurs, se rapprochant fortement de 0 et non de 1 montre un résultat de mauvaise adéquation.

Alors aux vues des mauvais résultats des différentes adéquations et des différentes chaînes de gènes, il est impossible de battre toutes les stratégies rassemblées.

b) Nous essayons donc de combattre seulement certaines stratégies, par groupe de stratégies.

- Premièrement nous essayons d'affronter automaton contre les stratégies gentilles:

[Automaton] *contre* [Gentle(), GentleSulky()] :

Avec un gène de longueur 21, l'adéquation est faible (0.357) en abandonnant
Avec un gène de longueur 21, l'adéquation est faible (0.357) sans abandonner

Ces valeurs, se rapprochant de 0 et non de 1 montre un résultat de moyenne adéquation.

Avec un gène de longueur 5, l'adéquation est faible (0.357) en abandonnant
Avec un gène de longueur 5, l'adéquation es faible (0.357) sans abandonner

Ces valeurs, se rapprochant de 0 et non de 1 montre un résultat de moyenne adéquation.

Les résultats d'adéquation sont identiques quelque soit le gène, ou quelque soit l'abandon ou non. Alors pour aller plus vite nous cherchons directement dans les chaînes de gène de 5 pour la suite.

- Deuxièmement, nous essayons d'affronter automaton contre les stratégies méchantes:

[Automaton] *contre* [ Bad(), BadSulky()]

Avec un gène de longueur 5, l'adéquation est faible (0.3514) en abandonnant
Avec un gène de longueur 5, l'adéquation est faible (0.3519) sans abandonner

Ces valeurs, se rapprochant de 0 et non de 1 montre un résultat de moyenne adéquation.
De plus nous retrouvons approximativement les mêmes résultats que pour les stratégies gentilles (environ 0.35)

- Troisièmement, nous essayons d'affronter automaton contre les stratégies de types Sulky:

[Automaton] *contre* [ GentleSulky(), BadSulky(), FoolSulky()]

Avec un gène de longueur 5, l'adéquation est faible (0.3571) en abandonnant
Avec un gène de longueur 5, l'adéquation est faible (0.238) sans abandonner

Ces valeurs, se rapprochant de 0 et non de 1 montre un résultat de moyenne adéquation. De plus, dans ce cas là, l'abandon obtient une meilleure adéquation, donc assurera un score maximal.

- Quatrièmement, nous essayons d'affronter automaton contre les stratégies qui modifient leur comportement en fonction de la stratégie adverse:

[Automaton] *contre* [ Tit4Tat(), WinStayLooseShift(),Pavlov()]

Avec un gène de longueur 5, l'adéquation est faible (0.2444) en abandonnant
Avec un gène de longueur 5, l'adéquation est faible (0.2498) sans abandonner

Ces valeurs, se rapprochant de 0 et non de 1 montre un résultat de moyenne adéquation. De plus, dans ce cas là, à quelque millième près il vaut mieux ne pas abandonner afin d'obtenir un score maximal.

# Conclusion

Nous constatons donc que la stratégie automaton ne peut pas être viable. Même en recherchant les meilleurs utilisations de cette stratégie elle ne se montre pas efficace avec une adéquation maximale à 0,35 alors qu'une adéquation parfaite serait à 1. Elle ne gagne donc même pas la moitié des points. Il est à noter que la stratégie automaton n'est pas apprenante, c'est à dire qu'elle n'essaie pas d'analyser la manière de jouer de son adversaire et de prédire le prochain coup. Elle joue toujours la même chose quelque soit son résultat. Sur des recherches futurs comparer les stratégies apprenantes entre elles grâce à l'algorithme génétique. Nous pourrions aussi chercher quelle est la meilleure stratégie Automaton à l'aide d'une stratégie apprenante, autrement dit trouver le meilleur système de règles grâce à la stratégie apprenante tel que Shannon.