Théorie des jeux algorithmique – TP

Dudo

Dernière modification: 20 janvier 2023

L'objectif de ce TP est de calculer un équilibre de Nash dans un jeu sous forme extensive à information imparfaite et à somme nulle : *Dudo*. On pourra retrouver les règles complètes sur la page Wikipedia https://fr.wikipedia.org/wiki/Dudo.

Il s'agit d'un jeu de bluff, qui peut se joueur à 2 joueurs et plus. De nombreuses variantes existent ; on considère ici le principe suivant : chaque joueur a initialement 5 dés à 6 faces qu'il fait rouler et consulte secrètement. À partir de ses résultat, à son tour, chacun fait une enchère concernant le nombre de dés total d'une certaine valeur en montant sur l'enchère précédente, ou annonce *dudo* impliquant que l'enchère précédente n'est pas réalisée. On vérifie alors en consultant tous les dés. Si l'enchère était réalisée, le joueur annonçant *dudo* perd un dé, sinon c'est celui qui avait fait l'enchère non réalisée qui perd un dé. Puis on recommence, jusqu'à ce qu'un seul joueur ait encore au moins un dé : c'est le gagnant de la partie.

Les enchères indiquent le nombre **minimal** de dés d'une certaine valeur : cinq fois 4, ou sept fois 6, etc. Une enchère est supérieure à une autre si elle concerne le même nombre de dés mais une valeur plus grande, ou plus de dés. La valeur 1 est particulière car elle compte pour toutes les enchères. Ainsi cinq fois 4 signifie en fait un nombre total de dés de valeur 1 **ou** 4 supérieur ou égal à cinq.

Une enchère portant sur le nombre de 1 est donc plus forte que sur tous les autres valeurs (pour un même nombre de dés). On se référera à la règle complète pour le calcul exact de la valeur des enchères.

Dans notre cas, pour simplifier, on considère 2 joueurs possédant chacun un seul dé : c'est une situation de fin de partie. Le jeu se fait donc sur une seule manche puisque celui qui perd n'a plus de dé. Par ailleurs, il n'y a avec deux dés que 12 enchères possibles. Par ordre croissant : 1×2, 1×3, 1×4, 1×5, 1×6, 1×1, 2×2, 2×3, 2×4, 2×5, 2×6, 2×1. Et bien sûr, on peut annoncer *dudo*.

Pour résoudre ce jeu, on va utiliser l'algorithme CFR vu en cours. La valeur de la stratégie optimale pour le joueur 1 qui commence est $\frac{-7}{258}$ soit environ $-0,027$.

On représente un *état* du jeu par un triplet (d_1, d_2, L) où d_i est le dé du joueur i et L est la liste des annonces réalisées (dans leur ordre d'occurrence, incluant potentiellement *dudo*).

On encode les annonces par un entier entre 0 et 11 de sorte qu'elles soient classées par ordre de force (avec 11 la plus forte). On représente *dudo* par 12.

On suppose que c'est toujours le joueur 1 qui commence. À partir de la longueur de L, on peut donc retrouver quel est le joueur qui doit joueur dans un état donné.

Un *ensemble d'information* pour le joueur i sera un couple (d_i, L) où manque la valeur du dé de l'autre joueur. Chaque ensemble d'information contient donc 6 états. Chaque joueur

joue selon une stratégie comportementale, qui donne une distribution de probabilités sur les coups possibles dans chaque ensemble d'information.

Q1. Écrire une fonction `eval_claim` qui à partir d'un état donné dans lequel la dernière annonce est *dudo* (par l'un ou l'autre joueur) indique 1 sur le joueur 1 a raison et -1 si le joueur 2 a raison.

Q2. Écrire une fonction `legal_moves` qui à partir d'un état donné donne la listes des annonces possibles : on doit toujours monter sur l'annonce précédente et on ne peut pas annoncer *dudo* au premier tour.

Q3. Écrire une fonction récursive `eval_strategy_rec` qui a partir d'un état, et d'un profil de stratégies comportementales, calcule la valeur de l'état pour ce profil, c'est-à-dire l'espérance d'utilité pour le joueur qui doit jouer obtenue en suivant le profil de stratégies.

On suppose que la stratégie dans chaque ensemble d'information n'est pas nécessairement normalisée. Il faut donc la normaliser explicitement.

Q4. Dans *Dudo*, il n'y qu'un seul coup de *Chance*, au tout début, pour choisir la valeur des dés de chaque joueur. Écrire une fonction `eval_strategy` qui a partir d'un profil de stratégies comportemental calcule la valeur pour ce profil de la racine de l'arbre de jeu.

Notez que la valeur du profil de stratégies dans lequel les deux joueurs jouent uniformément au hasard est environ -0.0324.

On veut maintenant implémenter l'algorithme CFR. Il a un comportement structurellement un peu similaire à l'évaluation de la valeur de la stratégie. Les différences principales sont :

1. Au lieu d'évaluer une stratégie quelconque, on évalue la stratégie *regret matching* pour chaque joueur (Q7.) ;
2. On calcule en plus, pour un joueur i donné en paramètre, les regrets cumulés et la stratégie moyenne dans chacun de ses ensembles d'information (Q6.).
3. Pour cela on a besoin de connaître la contribution $\pi_i^s(h)$ du joueur i à la probabilité d'atteindre l'état courant, c'est-à-dire le produit des probabilités de choisir les actions qui relèvent de ce joueur dans l'historique h des actions menant à l'état courant. On aura similairement besoin de la probabilité contre-factuelle $\pi_{-i}^s(h)$ qui contient la contribution de l'autre joueur et de Chance. Ces deux probabilités sont calculées dans la Q5.

Q5. En vous inspirant de `eval_strategy_rec`, écrire une nouvelle fonction `cfr_rec` qui complète `eval_strategy_rec` en mémorisant en paramètres supplémentaires les contributions des deux joueurs ($\pi_i^s(h)$ et $\pi_{-i}^s(h)$) à la probabilité d'atteindre l'état courant h en suivant le profil de stratégies s.

On ignorera la contribution de *Chance* (qui irait normalement dans $\pi_{-i}^s(h)$) en traitant le premier coup à part dans la Q8.

Q6. Compléter `cfr_rec` avec un calcul postfixe pour le joueur i, dans les ensembles d'information dans lesquels il doit jouer, de la stratégie moyenne (qu'on laissera non-normalisée), et du regret cumulé pour chaque action possible. Ces deux objets sont donc des sommes de termes obtenus au travers appels successifs à `cfr_rec` avec des profils de stratégies différents. Chacun de ces appels successifs résulte d'un appel à `cfr` (Q7.) qui correspond intuitivement à la réalisation d'une partie d'entraînement.

On passera en plus un numéro de joueur en paramètre et on ne mettra à jour les regrets cumulés et la stratégie moyenne que lorsque ce joueur joue.

Q7. Supprimer dans `cfr_rec` la donnée du profil de stratégies. Le remplacer par un calcul de la stratégie *Regret matching* dans l'ensemble d'information qui contient l'état courant.

Q8. Écrire une fonction `cfr` qui implémente l'algorithme CFR : on réalise n itérations. Pour chacune, pour chacun des deux joueurs, on calcule la valeur de la stratégie regret matching à la racine de l'arbre de jeux à partir d'appels à `cfr_rec` pour chacun des états que peut choisir *Chance*. Les appelles à `cfr_rec` mettent à jour la stratégie moyenne et les regrets cumulés dans chaque ensemble d'information du joueur courant. À la fin de l'itération, on évalue la valeur de la stratégie moyenne. Le résultat doit converger vers -0.027 en environ 250 itérations.

Q9. Enregistrer la stratégie moyenne finale sur le disque dur [1].

Q10. Écrire une fonction `play` qui permet de jouer (répétitivement) contre une stratégie comportementale donnée. Tester avec la stratégie enregistrée précédemment. Arrivez-vous à battre la valeur optimale théorique sur un nombre significatif de parties ?

Q11. Ajouter l'annonce *calzo* qui indique qu'on est d'accord avec la dernière enchère. On gagne si le nombre de dés est **exactement** celui annoncé (ni moins, ni plus) ;

À partir de là, on pourrait calculer sur le même principe les stratégies pour 1 dé contre 2 dés, 2 dés contre 2 dés, etc. pour obtenir une IA pour le jeu complet.

1. En Python, on peut utiliser par exemple `pickle` pour sérialiser un dictionnaire.