La méthode des poids multiplicatifs : un algorithme générique pour l'apprentissage et l'optimisation

Richard Lassaigne

Parmi les différentes classes de méthodes de conception d'algorithmes, la méthode des poids multiplicatifs occupe une place particulière. Si l'on considère qu'une initiation à l'algorithmique doit être assez complète, on peut lui assigner deux objectifs. Le premier est de présenter des algorithmes efficaces pour des problèmes fondamentaux bien résolus, comme par exemple les problèmes de plus court chemin, de flot maximum, de coupure minimum ou de couplage dans les graphes, de programmation linéaire ainsi que leurs applications. Le second est de fournir des outils de classification des problèmes qui ne sont pas bien résolus et de mettre au point des approches permettant cependant de s'y attaquer :

- montrer qu'un grand nombre de problèmes intéressants dans les domaines de la combinatoire, des graphes ou celui de l'optimisation sont *NP-difficiles*, c'est-à-dire peu susceptibles de donner lieu à des algorithmes de résolution exacte efficaces,
- étudier les problèmes de *décision en ligne*, comme ceux posés dans le cadre des jeux ou celui de l'apprentissage,
- utiliser des algorithmes d'*approximation* ou *probabilistes*, pour lesquels on peut borner les paramètres d'erreur, lorsque cela est possible.

La méthode des poids multiplicatifs est une présentation simple, de manière unifiée, de différents algorithmes découverts indépendamment dans des domaines aussi divers que l'apprentissage, la théorie des jeux ou l'optimisation. Le cadre de cet algorithme est le suivant. Un preneur de *décision* a le choix entre n décisions et doit de manière répétitive prendre une décision et obtenir une *récompense* associée, ou payer un certain *coût*. L'objectif du preneur de décision, à long terme, est d'obtenir une récompense totale qui soit comparable à la récompense de la décision fixée qui *maximise* la récompense totale de manière rétrospective. Bien que cette meilleure décision ne puisse être connue a priori, il est cependant possible de réaliser cet objectif en maintenant des *poids* sur les différentes décisions et en choisissant les décisions de manière aléatoire, avec *probabilités proportionnelles* aux poids. A chaque étape de décision, les poids sont *mis à jour en* les multipliant par des facteurs qui dépendent de la récompense de la décision prise à cette étape. De manière intuitive, cette stratégie fonctionne car elle a tendance à concentrer des poids plus importants sur les décisions à récompenses plus fortes à long terme.

Parmi les précurseurs de la méthode des poids multiplicatifs, on peut d'abord citer George W. Brown qui proposa en théorie des jeux un algorithme nommé *jeu fictif* [2] : à chaque étape, chaque joueur observe les actions effectuées par son adversaire lors des étapes précédentes, met à jour ses croyances concernant les stratégies adverses et choisit la meilleure réponse relativement à ses croyances. Michael D. Grigoriadis et Leonid G. Khachiyan ont montré comment une variante *probabiliste* de jeu fictif [6] permet de résoudre les jeux à deux joueurs à somme nulle de manière efficace : c'est précisément la méthode des *poids multiplicatifs*. Dans le domaine de l'apprentissage, la plus ancienne forme de cette méthode est due à Nick Littlestone pour son algorithme de Winnow [7]. Cet algorithme fut généralisé par Nick Littlestone et Manfred K. Warmuth sous la forme de l'algorithme de majorité pondérée [8] et plus tard par Yoav Freud et Robert E. Schapire sous la forme de l'algorithme de Hedge [4] dans le cadre de l'apprentissage en ligne. Les deux derniers auteurs adaptèrent également la méthode pour la résolution *adaptative* des jeux [5].

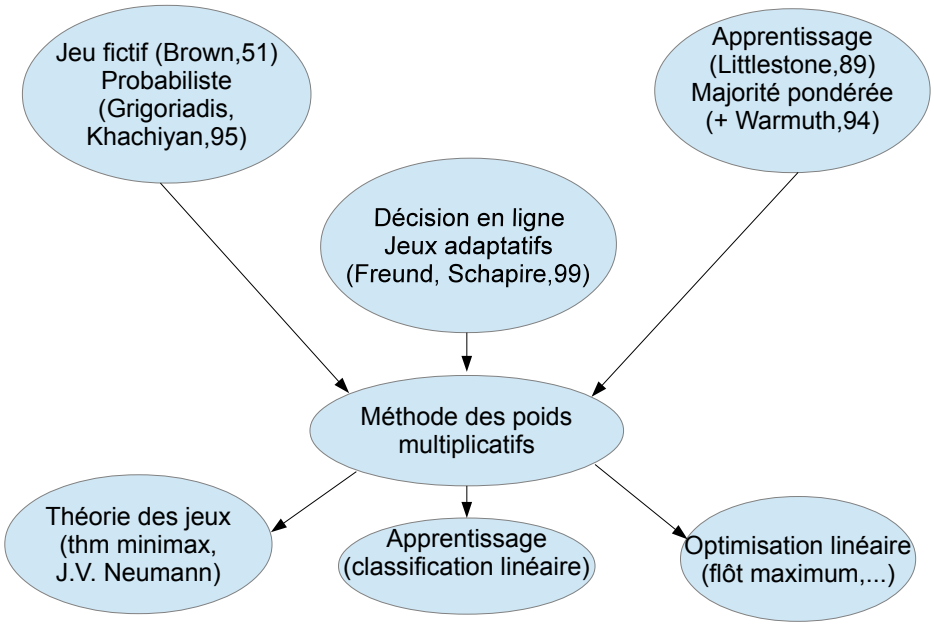

FIGURE 1 – la méthode des poids multiplicatifs

Les principaux auteurs d'une présentation unifiée de la méthode sont Sanjeev Arora, Elad Kazan et Satyen Kale dans leur fameux article[1] qui réalisait un état de l'art sur le sujet. Sur le plan pédagogique, les cours de Tim Roughgarden à Stanford sur l'algorithmique en général et sur la théorie algorithmique des jeux sont une excellente introduction à cette méthode. Enfin, il est difficile de parler d'algorithmes et de complexité sans citer Christos Papadimitriou et ses contributions à la *théorie algorithmique des jeux*. En particulier, il a étudié l'applicabilité de la méthode des poids multiplicatifs à différentes classes de jeux [3], ainsi que, récemment, à la théorie de l'évolution.

La première partie de l'article expose la problématique de la prise de décision en ligne et définit les critères de performances d'un tel algorithme à l'aide de la notion de *regret* relativement à la meilleure suite d'actions ou à la meilleure action, de manière rétrospective. La deuxième partie présente la méthode des *poids multiplicatifs*. On utilise une distribution de probabilités sur les actions où la probabilité de chaque action est proportionnelle à un certain poids. Les poids sont mis à jour de manière multiplicative en tenant compte des récompenses espérées de manière à obtenir une majoration efficace du regret. La troisième partie est consacrée à la présentation de quelques exemples significatifs, mais non exhaustifs, d'application de la méthode. Dans le cadre de la théorie des jeux, on peut obtenir une version constructive du *théorème minimax* pour les jeux à somme nulle. Le deuxième exemple est l'algorithme de Winnow qui est un algorithme d'apprentissage en ligne pour le problème de *classification linéaire*. La dernière application présente une méthodologie d'approximation pour certains problèmes de *programmation linéaire*, dans le cas de systèmes de très grande taille.

1 Les problèmes de décision en ligne

Un problème de décision en ligne est caractérisé par le fait que l'entrée arrive élément par élément et qu'un algorithme prend une décision irrévocable à chaque étape où il reçoit un nouvel élément. Par exemple, dans un problème d'ordonnancement de tâches, on considère souvent que les tâches arrivent en ligne.

1.1 Le modèle de décision en ligne

Le modèle de prise de décision en ligne est défini par un ensemble A de $n \geq 2$ actions et un horizon de temps $T \geq 1$: à chaque étape de temps $t = 1, 2, \ldots, T$

- un preneur de décision choisit en ligne une *distribution de probabilités* p_t sur les actions,
- un adversaire choisit un vecteur de *récompenses* $r_t : A \longrightarrow [-1, +1]$,

- le preneur de décision choisit une *action* a_t suivant la distribution p_t et reçoit la récompense $r_t(a_t)$,
- le preneur de décision *apprend* le vecteur de récompenses r_t.

Un *algorithme* de décision en ligne spécifie à chaque étape t la distribution de probabilités p_t comme une fonction des vecteurs de récompenses $r_1, \ldots, r_{t-1}$ et des actions réalisées $a_1, \ldots, a_{t-1}$. Un *adversaire* pour un tel algorithme $\mathcal{A}$ spécifie à chaque étape le vecteur de récompenses r_t comme une fonction des distributions de probabilités $p_1, \ldots, p_t$ utilisées par $\mathcal{A}$ et des actions réalisées $a_1, \ldots, a_{t-1}$.

1.2 Quels sont les objectifs possibles pour un algorithme en ligne ?

La situation ne semble pas équitable : l'adversaire peut choisir le vecteur de récompenses r_t après le choix de la distribution p_t par le preneur de décision. Quelle garantie peut-on espérer pour un bon algorithme en ligne ? Une première idée serait que la récompense totale espérée soit proche de la récompense de la meilleure suite d'actions rétrospectivement. L'objectif $\sum_{t=1}^{T} max_{a \in A} r_t(a)$ est trop fort : un exemple simple montre qu'il n'y a pas d'espoir d'atteindre une récompense proche de celle de la meilleure suite d'actions rétrospectivement. On suppose $A = \{1, 2\}$ et on considère un algorithme de décision en ligne arbitraire. A chaque étape,l'adversaire choisit le vecteur de récompenses de la manière suivante : si l'algorithme choisit une distribution p_t telle que la probabilité de l'action 1 est au moins $1/2$, l'adversaire choisit le vecteur $r_t = \{-1, 1\}$, sinon le vecteur $r_t = \{1, -1\}$. L'adversaire force ainsi la récompense espérée de l'algorithme à être négative, alors que la récompense de la meilleure suite d'actions est T.

Plutôt que de comparer la récompense totale espérée à celle de la meilleure suite d'actions, on va la comparer à la récompense qu'obtiendrait la meilleure action fixée rétrospectivement. L'objectif devient ainsi : $max_{a \in A} \sum_{t=1}^{T} r_t(a)$. La notion de *regret* permet de mesurer le performance de l'algorithme relativement à un optimum raisonnable.

Définition 1 *Etant donné des vecteurs de récompenses $r_1, \ldots, r_T$, le regret de la suite d'actions $a_1, \ldots, a_T$ est :*

$$\underbrace{max_{a \in A} \sum_{t=1}^{T} r_t(a)}_{\text{meilleure action fixée}} - \underbrace{\sum_{t=1}^{T} r_t(a_t)}_{\text{algorithme}}$$

Un bon algorithme en ligne serait un algorithme permettant de rendre le regret aussi proche de 0 que possible. On remarque que dans le cas de deux actions avec un vecteur de récompenses $r_t = \{1, -1\}$, le regret dans le pire des cas est $2T$. Dans la suite, un regret linéaire en T sera considéré comme mauvais.

Quelle est la justification pour l'objectif de la meilleure action fixée rétrospectivement ? D'abord, il existe des algorithmes simples et naturels d'apprentissage dont les performances sont compétitives. Ensuite, réaliser cet objectif n'est pas évident. Enfin, cette mesure de performances est suffisante pour obtenir de nombreuses applications intéressantes, comme on le verra dans la dernière partie. Un exemple naturel d'algorithme de décision en ligne est de *suivre le leader* : à chaque étape t, on choisit l'action a avec la récompense cumulée $\sum_{u=1}^{t-1} r_t(a)$ maximum. L'exemple suivant montre que cet algorithme et, plus généralement, tout algorithme déterministe, peut avoir un regret qui croît linéairement avec T. On considère un algorithme déterministe qui, à chaque étape t, choisit une seule action a_t. Une stratégie évidente pour l'adversaire consiste à mettre la récompense de cette action à 0 et celle de toute autre action à 1. La récompense cumulée de l'algorithme est alors 0 alors que celle de la meilleure action rétrospectivement est au moins $T(1-\frac{1}{n})$. Même dans le cas de deux actions, le regret dans le pire des cas est au moins $\frac{T}{2}$.

La question se pose de savoir s'il existe des algorithmes de décision en ligne pour lesquels le regret soit sous-linéaire relativement à l'horizon de temps T. Pour les *algorithmes probabilistes*, l'exemple suivant limite le taux de décroissance du regret en fonction de l'horizon de temps. On suppose que le nombre d'actions est $n = 2$ et que l'adversaire choisit chaque vecteur de récompenses r_t de manière indépendante et uniforme entre $(1, -1)$ et $(-1, 1)$. De quelque manière astucieuse ou bête que fonctionne un algorithme de décision en ligne, sa récompense espérée à chaque étape, relativement à ce choix aléatoire de vecteur de récompenses, est exactement 0 ainsi que sa récompense cumulée espérée. La récompense cumulée espérée de la meilleure action rétrospectivement est $b\sqrt{T}$, où b est une constante indépendante de T. Ceci est une conséquence du fait que si on lance en l'air une pièce équitable T fois, le nombre de *face* espéré est $\frac{T}{2}$ et la déviation standard est $\frac{1}{2}\sqrt{T}$.

Etant donné un algorithme de décision en ligne $\mathcal{A}$, un choix aléatoire de vecteur de récompenses force $\mathcal{A}$ à avoir un regret espéré d'au moins $b\sqrt{T}$, où l'espérance est relative au choix du vecteur de récompenses et aux actions réalisées. Au moins un choix de vecteur de récompenses induit un adversaire qui force $\mathcal{A}$ à avoir un regret espéré d'au moins $b\sqrt{T}$, où l'espérance est relative aux actions réalisées. Un argument similaire montre que, avec n actions, le regret espéré d'un algorithme de décision en ligne ne peut croître plus lentement que $b\sqrt{T \ln n}$, où b est une constante indépendante de n et T.

2 Algorithme des poids multiplicatifs

Dans cette section, on présente un algorithme probabiliste simple et naturel avec un regret espéré optimal dans le pire des cas, de l'ordre de la *borne inférieure* précédente.

2.1 Résultats et conception de l'algorithme

Théorème 1 *Il existe un algorithme de décision en ligne qui, pour tout adversaire, a un regret espéré d'au plus* $2\sqrt{T\ ln\ n}$.

Une conséquence immédiate est que le nombre d'étapes nécessaire pour rendre le regret espéré en moyenne (sur le temps) aussi petit que l'on veut est seulement *logarithmique* dans le nombre d'actions.

Corollaire 1 *ll existe un algorithme de décision en ligne qui, pour tout adversaire et tout* $\varepsilon > 0$*, a un regret espéré en moyenne d'au plus* ε *après au plus* $(4\ ln\ n)/\varepsilon^2$ *étapes.*

C'est cette forme de garantie qui sera utilisée dans les applications. Les principes de conception de l'algorithme des poids multiplicatifs, noté MW, sont les suivants :

1. les performances passées des actions guident le choix d'une action à chaque étape et la probabilité de choix d'une action croît avec sa récompense cumulée,
2. la probabilité de choix d'une action peu performante doit décroître de manière exponentielle.

Le premier principe est essentiel pour obtenir un regret sous-linéaire en T et le second pour des bornes optimales sur le regret. L'algorithme MW maintient un poids, coefficient de crédibilité, pour chaque action. A chaque étape t, l'algorithme choisit une action a avec probabilité $p_t(a)$ proportionnelle au poids actuel $w_t(a)$. Le poids de chaque action évolue dans le temps selon les performances passées de l'action. La distribution de probabilités sur les actions à l'étape t est notée $\mathbf{p_t}$.

Algorithme des poids multiplicatifs (MW) :
Initialisation : $w_1(a) = 1$ pour tout $a \in A$
Pour toute étape $t = 1, 2, \ldots, T$
- utiliser la distribution $\mathbf{p_t} = \mathbf{w_t}/\Gamma_t$ sur les actions
  où $\Gamma_t = \sum_{a \in A} w_t(a)$ est la somme des poids
- étant donné le vecteur de récompenses $\mathbf{r_t}$, pour toute action $a \in A$,
  mettre à jour le poids $w_{t+1}(a) = w_t(a)(1 + \eta r_t(a))$

Le paramètre η est compris entre 0 et 1/2 et est choisi à la fin de la preuve du théorème. Lorsque le paramètre η est proche de 0, la distribution $\mathbf{p_t}$ est proche de la distribution uniforme. Les petites valeurs de η encouragent l'*exploration* et les grandes valeurs l'*exploitation*, dans l'esprit de l'algorithme *suivre le leader*. Le paramètre η est un bouton permettant une interpolation entre ces deux extrêmes. L'algorithme des poids multiplicatifs est très facile à implanter, puisque la seule condition à remplir est la mise à jour des poids à chaque étape.

2.2 Correction de l'algorithme MW

Le problème posé par la preuve de correction est de savoir comment mettre en relation les deux quantités intervenant dans la définition du regret qui n'ont apparemment pas grand chose à voir ensemble : la récompense espérée de l'algorithme MW et la récompense espérée de la meilleure action fixée. L'idée de la preuve est de relier ces deux quantités à une quantité intermédiaire, la *fonction potentiel*, dont la valeur finale est la somme $\Gamma_{T+1} = \sum_{a \in A} w_{T+1}(a)$ des poids des actions en fin d'exécution de l'algorithme.

La preuve de correction s'effectue en deux étapes. La première montre comment la somme des poids évolue dans le temps. La récompense espérée γ_t à l'étape t est :

$$\gamma_t = \sum_{a \in A} p_t(a).r_t(a) = \sum_{a \in A} \frac{w_t(a)}{\Gamma_t}.r_t(a)$$

On calcule l'évolution de la fonction potentiel à l'étape t :

$$\Gamma_{t+1} = \sum_{a \in A} w_{t+1}(a) = \sum_{a \in A} w_t(a)(1 + \eta r_t(a))$$

$$\Gamma_{t+1} = \Gamma_t(1 + \eta.\gamma_t)$$

A chaque étape, on peut borner la fonction potentiel : $\Gamma_{t+1} \leq \Gamma_t.e^{\eta.\gamma_t}$ On obtient ainsi une borne supérieure finale pour la fonction potentiel :

$$\Gamma_{T+1} \leq \Gamma_1 \prod_{t=1}^{T} e^{\eta.\gamma_t} = n.e^{\eta \sum_{t=1}^{T} \gamma_t}$$

La récompense totale espérée $\sum_{t=1}^{T} \gamma_t$ est bornée inférieurement par une fonction relativement simple de la valeur finale de la fonction potentiel.

La seconde étape consiste à montrer que s'il existe une meilleure action fixée, alors le poids de cette action permet de montrer que la valeur finale de la fonction potentiel est plutôt grande. Soit $OPT = \sum_{t=1}^{T} r_t(a^*)$ la récompense cumulée d'une meilleure action fixée a^*. On peut minorer la valeur Γ_{T+1} :

$$\Gamma_{T+1} \geq w_{T+1}(a^*) = \underbrace{w_1(a^*)}_{=1} \prod_{t=1}^{T} (1 + \eta r_t(a^*))$$

On utilise le fait : $\ln(1+x) \geq x - x^2$ pour $\mid x \mid \leq \frac{1}{2}$. Puisque que $\eta \leq \frac{1}{2}$ et $\mid r_t(a^*) \mid \leq 1$, la borne inférieure obtenue est alors :

$$\Gamma_{T+1} \geq \prod_{t=1}^{T} e^{\eta r_t(a^*) - \eta^2 (r_t(a^*))^2} = e^{\sum_{t=1}^{T} (\eta r_t(a^*) - \eta^2 (r_t(a^*)^2)}$$

$$\Gamma_{T+1} \geq e^{\eta OPT - \eta^2 T} \text{ avec } (r_t(a^*))^2 \leq 1 \text{ pour tout } t$$

En combinant avec la borne obtenue lors de la première étape, on a réussi à relier le regret espéré de l'algorithme avec le regret de la meilleure action fixée par l'intermédiaire de la valeur finale de la fonction potentiel :

$$n.e^{\eta \sum_{t=1}^{T} \gamma_t} \geq \Gamma_{T+1} \geq e^{\eta OPT - \eta^2 T}$$

En prenant le logarithme des deux côtés et en divisant par η :

$$\sum_{t=1}^{T} \gamma_t \geq OPT - \eta T - \frac{\ln \text{ n}}{\eta}$$

On choisit le paramètre η de manière à rendre égaux les deux termes d'erreur :
$\eta = \min(\sqrt{\ln \text{ n}/T}, 1/2)$. Le regret espéré de l'algorithme MW est donc en $O(\sqrt{T \ln \text{ n}})$:

$$OPT - \sum_{t=1}^{T} \gamma_t \leq 2\sqrt{T \ln \text{ n}}$$

On remarque que la *borne inférieure* sur le regret espéré d'un algorithme de décision en ligne est atteinte. D'autre part, le paramètre η suppose la connaissance de l'horizon de temps T. Des modifications mineures permettent d'étendre l'algorithme MW et la garantie sur le regret dans le cas d'un horizon T inconnu à priori, avec la constante 2 de

la borne remplacée par une constante plus grande, mais raisonnable. Enfin, pour obtenir le corollaire sur le regret espéré en moyenne, il suffit de diviser par T :

$$\frac{1}{T}(OPT - \sum_{t=1}^{T} \gamma_t) \leq 2\sqrt{\frac{\ln\ n}{T}}$$

Pour un horizon de temps $T = \frac{4\ln\ n}{\varepsilon^2}$, le regret espéré en moyenne est plus petit que ε. Un nombre d'étapes *logarithmique* dans le nombre d'actions permet de rendre le regret espéré en moyenne aussi petit que l'on veut.

2.3 Extensions de l'algorithme MW

Ce paragraphe est consacré à deux extensions de l'algorithme MW, très utiles pour les applications. La première extension concerne l'espérance du regret qui est calculée relativement au choix aléatoire de l'action à chaque étape. Le résultat du théorème s'applique non seulement par rapport à la meilleure action fixée rétrospectivement, mais aussi à la *meilleure distribution* de probabilités fixée sur les actions. En effet, la meilleure action fixée est au moins aussi bonne que la meilleur distribution fixée : pour toute distribution $\mathbf{p}$ sur A,

$$\sum_{t=1}^{T}\sum_{a\in A} p_a.r_t(a) = \sum_{a\in A} p_a(\sum_{t=1}^{T} r_t(a)) \leq \sum_{a\in A} p_a(max_{a\in A}\sum_{t=1}^{T} r_t(a)) \leq max_{a\in A}\sum_{t=1}^{T} r_t(a)$$

La seconde extension concerne l'ensemble des valeurs de *récompenses* que l'on fait passer de $[-1, 1]$ à $[-M, M]$. On effectue un changement d'échelle permettant d'obtenir un regret en moyenne $\leq \varepsilon$:

- on divise toutes les récompenses par M,
- on applique l'algorithme MW pour que le regret en moyenne soit $\leq \varepsilon/M$,
- on choisit l'horizon de temps $T = \frac{4M^2\ln\ n}{\varepsilon^2}$.

3 Jeux, apprentissage et optimisation

La dernière section présente différents contextes d'application de la méthode des poids multiplicatifs : la théorie des jeux, l'apprentissage et les problèmes d'optimisation.

3.1 Jeux à somme nulle et théorème minimax

Un jeu à somme nulle est défini par une matrice $A = (a_{ij})$ de taille $m \times n$ à valeurs réelles. Le joueur 1 choisit une ligne et le joueur 2 une colonne. L'élément a_{ij} de la matrice

A est le gain du joueur 1 lorsqu'il choisit la ligne i et que le joueur 2 choisit la colonne j. Le gain du joueur 2 dans ce cas est $-a_{ij}$, d'où le nom de jeu à somme nulle. Les lignes, respectivement les colonnes, sont aussi appelées *stratégies* (pures) du joueur 1, respectivement du joueur 2. Les joueurs peuvent aussi utiliser des stratégies *mixtes*, c'est-à-dire des distributions de probabilités sur les lignes, respectivement les colonnes. Par exemple, la matrice suivante définit le jeu bien connu Pierre-Papier-Ciseaux.

Joueur 1 \ Joueur 2	Pierre	Papier	Ciseaux
Pierre	0	-1	1
Papier	1	0	-1
Ciseaux	-1	1	0

Le gain espéré du joueur 1 lorsque la stratégie mixte du joueur 1 est $\mathbf{x}$ et la stratégie mixte du joueur 2 est $\mathbf{y}$ peut s'écrire :

$$\sum_{i=1}^{m}\sum_{j=1}^{n} Prob[(i,j)\ choisi]a_{ij} = \sum_{i=1}^{m}\sum_{j=1}^{n} Prob[i\ choisi].Prob[j\ choisi]a_{ij} = \mathbf{x}^T\mathbf{A}\mathbf{y}$$

Dans un jeu à deux joueurs et à somme nulle, est-il préférable pour un joueur d'engager une stratégie mixte avant ou après l'autre joueur ? Intuitivement, il y a un désavantage pour le premier joueur puisque le second peut s'adapter à la stratégie du premier. Le théorème minimax affirme que cela n'a pas d'importance.

Théorème 2 *(**Théorème minimax**) Pour tout jeu à deux joueurs et à somme nulle* $\mathbf{A}$*,*

$$max_{\mathbf{x}}(min_{\mathbf{y}}\mathbf{x}^T\mathbf{A}\mathbf{y}) = min_{\mathbf{y}}(max_{\mathbf{x}}\mathbf{x}^T\mathbf{A}\mathbf{y})$$

Dans le premier scénario, le joueur 2 joue de manière optimale par rapport à la stratégie du joueur 1 et le joueur 1 joue de manière optimale en anticipant la réponse du joueur 2. Dans le second scénario, les rôles sont inversés. Le théorème affirme que dans un jeu optimal, le gain espéré de chaque joueur est le même dans les deux scénarios. La preuve originelle de John von Neumann (1928) était *non constructive* et reposait sur un résultat de point fixe dû à Brouwer. Dans les années 1940, von Neumann a donné une deuxième preuve utilisant des arguments équivalents à la dualité forte en programmation linéaire. Dans la suite, le résultat du théorème minimax est obtenu à partir de la garantie assurée par l'algorithme des poids multiplicatifs. Etant donné un jeu à somme nulle défini par une matrice $\mathbf{A}$ à m lignes, dont les valeurs appartiennent à $[-M, M]$, et un paramètre $\varepsilon > 0$, on applique l'algorithme MW au cours de l'expérience suivante :

A chaque étape $t = 1, 2, \ldots, T = \frac{4M^2 \ln \mathrm{m}}{\varepsilon^2}$
- le joueur 1 choisit une stratégie mixte $\mathbf{p_t}$ utilisant l'algorithme **MW** où les actions sont les lignes,
- le joueur 2 répond de manière optimale avec la stratégie déterministe $\mathbf{q_t}$,
- si le joueur 2 choisit la colonne j, alors prendre comme récompense $r_t(i) = (a_{ij})$ pour toute ligne i, et alimenter l'algorithme **MW** avec le vecteur $\mathbf{r_t}$.

On remarque que le joueur 2 n'a jamais besoin de jouer aléatoirement : il choisit la colonne avec le meilleur gain espéré relativement à la stratégie mixte du joueur 1. On montre d'abord que le gain négatif espéré en moyenne du joueur 2 est au plus :

$$\max_{\mathbf{p}}(\min_{\mathbf{q}} \mathbf{p}^T \mathbf{A} \mathbf{q})$$

Le joueur 2 peut adapter ses meilleures réponses à chaque étape à la stratégie mixte du joueur 1. Soit $\hat{\mathbf{p}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{p}_t$ la stratégie ligne en moyenne et $\mathbf{q}^*$ une réponse optimale à $\hat{\mathbf{p}}$:

$$\max_{\mathbf{p}}(\min_{\mathbf{q}} \mathbf{p}^T \mathbf{A} \mathbf{q}) \geq \min_{\mathbf{q}} \hat{\mathbf{p}}^T \mathbf{A} \mathbf{q} = \hat{\mathbf{p}}^T \mathbf{A} \mathbf{q}^* = \frac{1}{T} \sum_{t=1}^{T} (\mathbf{p}_t)^T \mathbf{A} \mathbf{q}^*$$

$$\max_{\mathbf{p}}(\min_{\mathbf{q}} \mathbf{p}^T \mathbf{A} \mathbf{q}) \geq \frac{1}{T} \sum_{t=1}^{T} (\mathbf{p}_t)^T \mathbf{A} \mathbf{q}_t$$

La dernière inégalité est une conséquence du fait que $\mathbf{q}_t$ est une réponse optimale à $\mathbf{p}_t$ pour chaque étape t et le dernier terme représente bien le gain négatif espéré en moyenne du joueur 2 durant cette expérience. On montre ensuite que le gain espéré en moyenne du joueur 1 est au moins :

$$\min_{\mathbf{q}}(\max_{\mathbf{p}} \mathbf{p}^T \mathbf{A} \mathbf{q}) - \varepsilon$$

Soit $\hat{\mathbf{q}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{q}_t$ la stratégie colonne en moyenne. La garantie de l'algorithme **MW** assure que le gain espéré en moyenne du joueur 1 est au moins, à ε près, le gain espéré en moyenne obtenu par une stratégie mixte fixée $\mathbf{p}$:

$$\frac{1}{T} \sum_{t=1}^{T} (\mathbf{p}_t)^T \mathbf{A} \mathbf{q}_t \geq \max_{\mathbf{p}}(\frac{1}{T} \sum_{t=1}^{T} \mathbf{p}^T \mathbf{A} \mathbf{q}_t) - \varepsilon = \max_{\mathbf{p}} \mathbf{p}^T \mathbf{A} \hat{\mathbf{q}} - \varepsilon$$

$$\frac{1}{T} \sum_{t=1}^{T} (\mathbf{p}_t)^T \mathbf{A} \mathbf{q}_t \geq \min_{\mathbf{q}}(\max_{\mathbf{p}} \mathbf{p}^T \mathbf{A} \mathbf{q}) - \varepsilon$$

En faisant tendre ε vers 0, on obtient l'inégalité difficile du théorème minimax :

$$\max_{\mathbf{p}}(\min_{\mathbf{q}}\mathbf{p}^T\mathbf{A}\mathbf{q}) \geq \min_{\mathbf{q}}(\max_{\mathbf{p}}\mathbf{p}^T\mathbf{A}\mathbf{q})$$

L'inégalité dans l'autre sens est la partie facile du théorème puisqu'il est à priori moins bon de jouer en premier.

3.2 Un exemple d'apprentissage

Un problème de classification linéaire consiste, dans un espace de dimension d, à séparer un ensemble de données étiquetées positivement d'un ensemble de données étiquetées négativement par un hyperplan.

Etant donné m données étiquetées *positives* $\mathbf{x}^1, \ldots, \mathbf{x}^m \in \mathbb{R}^d$ et m' données étiquetées *négatives* $\mathbf{y}^1, \ldots, \mathbf{x}^{m'} \in \mathbb{R}^d$, le but est de calculer une fonction linéaire $h : \mathbb{R}^d \longrightarrow \mathbb{R}$ telle que :

$$h(\mathbf{z}) = \sum_{j=1}^{d} a_j z_j + b,\ h(\mathbf{x}^i) > 0 \text{ pour } i = 1, \ldots, m \text{ et } h(\mathbf{y}^i) < 0 \text{ pour } i = 1, \ldots, m'$$

On effectue d'abord quelques étapes de pré-calcul permettant de mettre le problème sous une forme plus pratique :

- on force $b = 0$ en ajoutant une $(d+1)$-ième variable avec $a_{d+1} = b$ et toute donnée a une nouvelle cordonnée égale à 1,
- on multiplie les coordonnées des données négatives $\mathbf{y}^i$ par -1 et le système de contraintes s'écrit alors $h(\mathbf{x}^i) > 0$ et $h(\mathbf{y}^i) > 0$,
- on peut demander que tous les coefficients a_j soient positifs en faisant 2 copies x_j^i et $-x_j^i$ de chaque coordonnée x_j^i et en interprétant les 2 coefficients correspondants a'_j et a''_j avec $a_j = a'_j - a''_j$ (on double alors la dimension $d+1$).

On suppose que le problème est résoluble avec une *marge d'erreur* $\varepsilon > 0$ c'est-à-dire qu'il existe un vecteur de coefficients $\mathbf{a}^* \in \mathbb{R}_+^d$ tel que :

$$\sum_{j=1}^{d} a_j^* = 1 \text{ et } \sum_{j=1}^{d} a_j^* x_j^i > \varepsilon \text{ pour toute donnée } \mathbf{x}^i$$

On remarque que s'il existe une solution au problème initial, alors il existe une solution *normalisée* satisfaisant la condition $\sum_{j=1}^{d} a_j^* = 1$. Etant donné une borne M sur la valeur absolue des données et $A = \{1, 2, \ldots, d\}$ l'ensemble des actions, on applique l'algorithme MW de la manière suivante :

A chaque étape $t = 1, 2, \ldots, T = \frac{4M^2 \ln d}{\varepsilon^2}$
- utiliser l'algorithme **MW** pour engendrer une distribution de probabilités $\mathbf{a}^t \in \mathbb{R}^d$,
- si $\sum_{j=1}^{d} a_j^t x_j^i > 0$ pour tout $\mathbf{x}^i$, alors stop et retourner $\mathbf{a}^t$,
- sinon choisir une donnée $\mathbf{x}^i$ avec $\sum_{j=1}^{d} a_j^t x_j^i < 0$ et définir un vecteur de récompenses $\mathbf{r}^t$ avec $r^t(j) = x_j^i$,
- fournir le vecteur $\mathbf{r}^t$ à l'algorithme **MW**.

L'analyse de l'algorithme montre qu'il fournit une solution convenable en au plus T étapes. Les vecteurs de récompenses sont tels que, à chaque étape t, le produit scalaire de $\mathbf{a}^t$ et $\mathbf{r}^t$ est négatif . En considérant $\mathbf{a}^t$ comme une distribution de probabilités, la récompense espérée est négative à chaque étape et la récompense espérée en moyenne est au plus 0. D'après l'hypothèse, il existe une distribution de probabilités $\mathbf{a}^*$ sur A telle que, à chaque étape t, la récompense espérée en choisissant $\mathbf{a}^*$ serait :

$$\sum_{j=1}^{d} a_j^* r^t(j) \geq \min_i \sum_{j=1}^{d} a_j^* x_j^i > \varepsilon$$

Tant que l'algorithme n'a pas trouvé de solution, le regret en moyenne est ε. L'algorithme **MW** garantit qu'après $T = \frac{4M^2(\ln d)}{\varepsilon^2}$ étapes, le regret en moyenne est au plus ε.

3.3 Problèmes d'optimisation linéaire

Dans ce paragraphe, on présente l'application de la méthode des poids multiplicatifs au cadre général des problèmes d'optimisation linéaire. Dans de nombreux cas, comme par exemple les problèmes de flot maximum, on connaît des algorithmes plus efficaces que la programmation linéaire. Cependant, la méthode des poids multiplicatifs peut être étendue aux problèmes de flot multi-produits, pour lesquels il n'existe pas d'algorithmes plus efficaces que la programmation linéaire. Le problème de base en optimisation linéaire est de vérifier la *faisabilité* du programme suivant :

$$\exists ? \mathbf{x} \in \mathcal{P} \quad \mathbf{A}\mathbf{x} \geq \mathbf{b} \quad (1)$$

où $\mathbf{A} \in \mathbb{R}^{m \times n}$ est une matrice de taille $m \times n$ et $\mathcal{P}$ est un ensemble convexe de $\mathbb{R}^n$,
- l'ensemble $\mathcal{P}$ représente les contraintes *faciles* à satisfaire,
- la matrice $\mathbf{A}$ les contraintes *difficiles* à satisfaire.

L'objectif est de concevoir un algorithme qui, étant donné un paramètre d'erreur ε,
- soit résout le problème avec une erreur additive $\leq \varepsilon$, c'est-à-dire trouve un $\mathbf{x} \in \mathcal{P}$ tel que pour tout i, $\mathbf{A}_i \mathbf{x} \geq b_i - \varepsilon$, où $\mathbf{A}_i$ est la ième ligne de la matrice $\mathbf{A}$,

- ou sinon prouve que le système n'est pas résoluble.

On suppose l'existence d'un algorithme, appelé *oracle* qui, étant donné un vecteur de probabilités $\mathbf{p}$ sur les m contraintes, résout le problème : $\exists ?\mathbf{x} \in \mathcal{P} \quad \mathbf{p}^T\mathbf{A}\mathbf{x} \geq \mathbf{p}^T\mathbf{b}$ (2). Une manière d'implanter cette procédure est de maximiser $\mathbf{p}^T\mathbf{A}\mathbf{x}$ sur $\mathbf{x} \in \mathcal{P}$. Il est raisonnable de supposer qu'une telle procédure existe puisqu'il suffit de vérifier la faisabilité d'une seule contrainte plutôt que de m contraintes. On remarque que si le problème (1) a une solution $\mathbf{x}^*$, alors la même solution satisfait aussi (2) pour tout vecteur de probabilités $\mathbf{p}$. Ainsi, s'il existe un vecteur $\mathbf{p}$ tel que aucun $\mathbf{x} \in \mathcal{P}$ ne satisfait (2), alors le problème originel n'est pas résoluble.

Définition 2 *Soit $M \geq 0$. Un oracle M-borné est un algorithme qui, étant donné un vecteur $\mathbf{p}$ résout le problème (2) avec pour tout i, $\mathbf{A}_i\mathbf{x} - b_i \in [-M, M]$.*

Le résultat suivant est obtenu par une application assez directe de la méthode des poids multiplicatifs. La même idée que dans le cas de la classification linéaire, à savoir associer un joueur aux *variables* de décision et un joueur aux *contraintes*, peut être utilisée dans l'approximation des programmes linéaires.

Théorème 3 *Soit $\varepsilon > 0$ un paramètre d'erreur. On suppose qu'il existe un oracle M-borné et que $M \geq \varepsilon/2$. Alors il existe un algorithme qui soit trouve un $\mathbf{x} \in \mathcal{P}$ tel que pour tout i, $\mathbf{A}_i\mathbf{x} \geq b_i - \varepsilon$, soit conclut que le système n'est pas résoluble. L'algorithme fait $O(\frac{M^2 log(m)}{\varepsilon^2})$ appels à l'oracle, avec un temps additionnel en $O(m)$ par appel.*

En conclusion, la méthode des poids multiplicatifs fournit un cadre général pour présenter des algorithmes d'*approximation* pour des problèmes très divers, mais pouvant se représenter sous la forme de problèmes de *décision en ligne*. La mise à jour des poids dans la distribution de probabilités sur les actions permet de diminuer de manière *exponentielle* l'influence des mauvaises stratégies. L'horizon de temps ne dépend que de manière *logarithmique* du nombre d'actions. Enfin, l'utilisation d'algorithmes probabilistes permet de réduire de manière drastique la *complexité en espace*.

Références

[1] Sanjeev Arora, Elad Hazan, and Satyen Kale. The Multiplicative Weights Update Method : a Meta-Algorithm and Applications. *Theory of Computing*, 8(1) :121–164, 2012.

[2] George W. Brown. Iterative solutions of games by fictitious play. In T. C. Koopmans, editor, *Analysis of Production and Allocation*, pages 374–376. Wiley, 1951.

[3] Constantinos Daskalakis, Rafael M. Frongillo, Christos H. Papadimitriou, George Pierrakos, and Gregory Valiant. On Learning Algorithms for Nash Equilibria. In *Algorithmic Game Theory - Third International Symposium, SAGT 2010, Athens, Greece, October 18-20, 2010. Proceedings*, pages 114–125, 2010.

[4] Yoav Freud and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer System Science*, 55(1) :119–139, 1997.

[5] Yoav Freud and Robert E. Schapire. Adaptative game playing using multiplicative weights. *Games and Economic Behavior*, 29 :79–103, 1999.

[6] Michael D. Grigoriadis and Leonid K. Khachiyan. A sublinear-time randomized approximation algorithm for matrix games. *Operations Resaerch Letters*, 18 :53–58, 1995.

[7] Nick Littlestone. *Mistake bounds and logarithmic linear-threshold learning algorithms*. PhD thesis, University of California at Santa Cruz, CA, USA, 1989.

[8] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2) :212–261, 1994.