Algorithmes en ligne

François Schwarzentruber

1 Location de ski

1€ = prix de la location pour un jour

b € = le prix de l'achat des skis, avec $b > 1$

1.1 Algorithme offline

— entrée : météo : une suite $x_1, \ldots, x_n$ avec $x_i \in \{beau, moche\}$
— sortie : une suite de décisions $d_i \in \{louer, acheter, skip\}$

Objectif : minimiser le coût, sachant que l'on veut skier tous les premiers beaux jours, et dès qu'un jour est moche, on ne skie pas et on part définitivement.

fonction algoOffline(x)
    $g :=$ nb de beaux jours consécutifs dans x
    si $g \leq b$ **alors** louer chaque jour **sinon** acheter le 1$^{\text{er}}$ jour

Proposition 1 L'optimum $\min(g, b)$ et l'algorithme offline est optimal.

1.2 Algorithme online déterministe

Il faut décider d_i avec un algorithme déterministe qui ne connaît que $x_{1..i}$.

fonction algoOnline()
    louer pendant les $b-1$ premiers jours
    acheter ensuite

Définition 2 (algorithme c-compétitif) Considérons un problème de minimisation. Un algorithme online ALG est c-compétitif s'il existe une constante b telle que pour toute instance x, on ait :

$$ALG(x) \leq c \times OPT(x) + b$$

où $OPT(x)$ le coût optimal pour x, et $ALG(x)$ le coût donné par l'algorithme ALG.

Définition 3 (ratio compétitif) Soit ALG un algorithme online de minimisation. Le ratio compétitif c_{ALG} est l'infimum sur les c où ALG est c-compétitif.

Proposition 4 Le ratio compétitif de l'algorithme online donné plus haut est de $2 - \frac{1}{b}$.

Démonstration.
Si $g < b$, $OPT(x) = ALG(x)$. Comme $(2 - \frac{1}{b}) \geq 1$, la borne est ok.
Sinon, si $g \geq b$, $ALG(x) = (b-1) + b$ alors que $OPT(x) = g$. On a $ALG(x) = b(2 - \frac{1}{b}) \leq g(2 - \frac{1}{b})$. En prenant $g = b$, on voit que la limsup est atteinte. ■

Théorème 5 On ne peut pas mieux faire avec un algorithme online déterministe.

Démonstration. Considérons un algorithme online optimal ALG^*. Les algorithmes online différent juste par le moment où ils décident d'acheter.
Si ALG^* décide d'acheter au jour $i \leq b-1$, alors il le fait aussi s'il fait moche au $i+1$-ème jour, puisque cette décision ne dépend pas du $i+1$-ème jour. Mais il aurait été alors préférable de continuer à louer au temps i.
Considérons $x =$ il fait beau i jour puis il fait moche.
$ALG^*(x) := i - 1 + b \geq i - 1 + i + 1 \geq 2i = 2OPT(x)$
Si ALG^* décide d'acheter au jour $i \geq b$,
$ALG^*(x) = i - 1 + b \geq b - 1 + b = (2 - \frac{1}{b})b = (2 - \frac{1}{b})OPT(x)$.
Résumé : l'algorithme ALG^* est ratio compétitif au moins $2 - \frac{1}{b}$. ■

1.3 Algorithme online randomisé

Il faut décider $d[i]$ avec un algorithme probabiliste qui ne connaît que $meteo[1..i]$.

```
fonction algoOnlineRandom()
    choisir aléatoirement i ∈ {0, b − 1} selon une distribution
    aléatoire donnée (p_0, ..., p_{b−1})
    louer pendant les i premiers jours
    acheter le i + 1-ème jour
```

Soit $p_i := \mathbb{P}(\text{choisir } i)$.

Définition 6 (algorithme c-compétitif) Considérons un problème de minimisation. Un algorithme online randomisé $ALGR$ est c-compétitif s'il existe une constante b telle que pour toute instance x, on ait :

$$\mathbb{E}(ALGR(x)) \leq c \times OPT(x) + b$$

où $OPT(x)$ est la valeur optimale pour x, et $ALGR(x)$ est la valeur retournée par l'algorithme $ALGR$.

Définition 7 (ratio compétitif) Pareil. Soit $ALGR$ un algorithme online de minimisation. Le ratio compétitif c_{ALGR} est l'infimum sur les c où $ALGR$ est c-compétitif.

Proposition 8 Le meilleur ratio compétitif de l'algorithme randomisé est $c := \frac{1}{1-(1-1/b)^b}$ pour $p_i = \frac{c}{b}(1-(1/b))^{b-1-i}$.

Démonstration. Par définition de l'espérance :

$$\mathbb{E}(\text{ALGR(x)}) = \sum_{i=0}^{b-1} \text{coût si } i \text{ est choisi} \times p_i$$

— Si $g < b$, $\mathbb{E}(\text{ALGR(x)}) = \sum_{i=0}^{g-1}(i+b)p_i + \sum_{i=g}^{b-1} gp_i$.
— Si $g \geq b$, $\mathbb{E}(\text{ALGR(x)}) = \sum_{i=0}^{b-1}(i+b)p_i = b + \sum_{i=0}^{b-1} ip_i$
On cherche donc à résoudre :

$$\text{minimiser } c \quad \begin{cases} \sum_{i=g}^{b-1} gp_i \leq cg \\ b + \sum_{i=0}^{b-1} ip_i \leq cb \\ \sum_{i=0}^{b-1} p_i = 1 \end{cases}$$

On peut vérifier que les p_i donnés dans l'énoncé de la proposition et c est bien une solution. C'est bien le minimum car on a égalité dans les inégalités-contraintes. ■

2 Robot aveugle

Un robot initialement en 0 se déplace sur la ligne $\mathbb{Z}$, qui contient une banane en d ou $-d$. But : trouver la banane en faisant le moins de mouvement. $OPT(x) = d$. L'algorithme offline : aller vers la banane.

Remarque 9 Seul l'information de la direction (gauche, ou droite) suffit pour avoir la performance de l'algorithme offline.

Cadre online : il ne voit rien sauf la case courante.

```
fonction algoOnline()
    phase 0 : se déplacer d'une case vers la droite et revenir en 0
    phase 1 : se déplacer de 2 cases vers la gauche et revenir en 0
    phase 2 : se déplacer de 4 cases vers la droite et revenir en 0
    ⋮
```

Proposition 10 Le ratio compétitif est de 9.

Démonstration. Le pire cas est quand la banane est à distance $d = 2^i + 1$ dans la direction $(-1)^i$.
La distance parcourue totale est :

$$ALG(x) = 2(1 + 2 + \ldots 2^{i+1}) + d$$

On a $ALG(x) = 22^{i+2} + d < 8d + d = 9d$.
Plus précisément, $\frac{ALG(x)}{OPT(x)} = 8\frac{d-1}{d} + 1$, donc oui, le ratio compétitif est bien de 9. ■

3 Arbre de Steiner

Définition 11 (arbre de Steiner) Soit un graphe complet $G = (V, E, c)$ complet pondéré positivement et vérifiant l'inégalité triangulaire ; des sommets $t_1, \ldots, t_k \in V$. Un *arbre de Steiner* est un arbre inclus dans G qui relie tous les sommets $t_1, \ldots, t_k$ entre eux et de poids minimal.

Définition 12 (terminaux) Les sommets $t_1, \ldots, t_k$ s'appellent des *terminaux*.

Définition 13 (Problème de l'arbre de Steiner online)
- entrée : un graphe complet $G = (V, E, c)$ complet pondéré positivement et vérifiant l'inégalité triangulaire ; des sommets $t_1, \ldots, t_k \in V$ donné un à un ;
- sortie : quand t_i arrive, il faut ajouter une arête à la solution courante pour que $\{t_1, \ldots, t_i\}$ soit de poids le plus petit possible.

```
fonction algoOnlineSteiner(G)
    T := ∅
    pour i := 1 à k faire
        quand t_i arrive, ajouter à T une arête (t_j, t_i) avec j < i qui est la plus légère
```

Théorème 14 L'algorithme est $2 \ln k$-compétitif, où k est le nombre de terminaux.

DÉMONSTRATION. Le lemme suivant permet de conclure via $cout(T) \leq \sum_{i=1}^{k} \frac{2opt}{i} \leq (2 \ln k) opt$.

Lemme 15 Pour tout $i = 1..k-1$, la i-ème arête la + chère dans la solution T est de coût $\leq \frac{2opt}{i}$ où opt est le coût d'un arbre de Steiner.

DÉMONSTRATION. Soit T^* un arbre de Steiner sur $t_1, \ldots, t_k$.
Soit $H = T^*$ où on ajoute une copie des arête de T^*.
H est un graphe eulérien. Soit C un tour eulérien de H.

$$cout(C) = \sum_{e \in C} c_e = 2opt$$

On définit le coût de connexion de t_i, noté $ccout(t_i)$, comme le coût de l'arête (t_j, t_i) ajouté par l'algorithme quand t_i est traité.

Soit $s_1, \ldots, s_k$ les terminaux ordonnés par $ccout$ décroissant. (c'est une permutation de $t_1, \ldots, t_k$). Le lemme consiste à démontrer que $ccout(s_i) \leq \frac{2opt}{i}$.

Soit $C_i = C$ où on prend des raccourcis pour ne visiter que $\{s_1, \ldots, s_i\}$. Comme C_i a exactement i arêtes, on a :

$$\min_{e \in C_i} c_e \leq \frac{1}{i} cout(C_i) \leq \frac{1}{i} cout(C) \leq \frac{2opt}{i}$$

Ainsi, il y a une arête (s_h, s_j) dans C_i de coût $\leq \frac{2opt}{i}$.
De s_h et s_j, on considère celui qui a arrive le plus tard dans l'ordre $t_1, \ldots, t_k$. Disons que c'est s_j.
Pendant le traitement de s_j, (s_h, s_j) est une option pour connecter s_j au reste de la solution.
Ainsi s_j est connecté au reste avec une arête de poids $\leq \frac{2opt}{i}$. Autrement dit, $ccout(s_j) \leq \frac{2opt}{i}$.
Comme $j < i$, et que s_i est après dans s_j dans la liste $s_1, \ldots, s_k$, $ccout(s_i) \leq \frac{2opt}{i}$.
■
■

4 Mise en cache

On considère un cache C de taille k. Le cache contient initialement des adresses. Si on ne trouve pas une donnée dans le cache, il y a *échec*. On peut décider de choisir une donnée du cache à supprimer et à remplacer par la nouvelle donnée. But : minimiser le nombre d'échec
- entrée : $X = x_1 x_2 ...$ où x_i est la i-ème adresse mémoire demandée
- sortie : décider quelles données remplacer

4.1 Algorithme offline optimal

En cas d'échec, supprimer/remplacer la page du cache qui apparaît le plus tard dans les demandes (ou jamais !)

On note $OPT(x)$ le nombre d'échecs de cet algorithme.

4.2 Algorithme online déterministe

```
fonction algoOnline()
    pour j = 1..n faire
        si x_j n'est pas dans le cache C alors
            choisir i ∈ {0, ..., k − 1}
            C[i] := x_j
```

Quelques stratégies pour le choix de i

LRU	Least Recently Used	remplacer la page qui n'a pas été utilisée pendant le plus longtemps
FIFO	First In first Out	remplacer la page qui a été le plus longtemps dans le cache
LIFO	Last In First Out	remplacer la page la plus récemment ajoutée au cache
LFU	Least Frequently Used	remplacer la page qui a été le moins utilisée

Proposition 16 L'algorithme online avec FIFO ou avec LRU est k-compétitif.

DÉMONSTRATION. On écrit $x = B_1 \dots B_T$ où :
— B_1 est le plus long préfixe de x contenant k adresses différentes.
— B_2 est le plus long préfixe de $x \setminus B_1$ contenant k adresses différentes.
— ...

Exemple 17 Considérons $x := 41215344123$.
On prend un cache de taille $k = 3$, le découpage est alors :
$x := [4121][5344][123]$.

Avec FIFO, il y a au plus k échecs dans chaque bloc : au plus k dans B_1 car le cache ne contient peut-être pas les bonnes pages initialement, au plus k dans B_2, etc. Il y a donc au plus kT échecs en tout.
$ALG(X) \leq Tk$
La première page de B_{i+1} est différente de toutes les pages de B_i (sinon elle serait dans B_i). Ainsi tout algorithme offline fait au moins $T - 1$ ratés. $OPT(X) \geq T - 1$.
Ainsi : On a

$$ALG(X) \leq Tk \leq (T-1)k + k \leq kOPT(X) + k.$$

■

Remarque 18 LRU marche mieux en FIFO. Et si k augmente, le ratio valant k augmente, alors que le cache fonctionne mieux. Pourquoi ?

4.3 Borne inférieure pour algorithme online déterministe

Proposition 19 S'il y a $k + 1$ pages, il n'y a pas de meilleur algorithme online : on ne peut pas faire mieux qu'un ratio compétitif de k.

DÉMONSTRATION. Soit A un algorithme online. Prenons $k+1$ pages. Si le cache est plein, l'adversaire choisit à chaque fois une page non présente. Echec à chaque étape. L'algorithme offline lui peut virer la page qui sera demandé dans trop longtemps, dans au moins $k - 1$ étapes. ■

4.4 Algorithme online randomisé : l'algorithme RMA

RMA = randomized marking algorithm

```
fonction RMA()
    C[1..k] := le tableau contenant les données du cache
    M[1..k] := [false, ..., false] (marques)
    pour j := 1 à n faire
        si x_j est présente dans C alors
            soit i avec C[i] = x_j
            M[i] := true
        sinon
            si pour tout i = 1..k, M[i] = true alors
                M[1..k] := [false, ..., false]
            S := {i = 1..k | M[i] = false}
            choisir uniformément i dans S
            C[i] := x_j
            M[i] := true
```

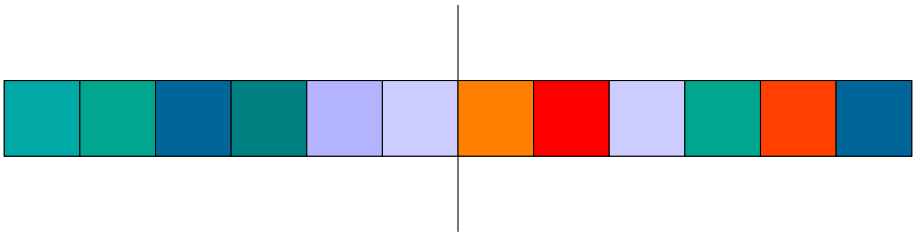
Définition 20 $H_k := \sum_{i=1}^{k} \frac{1}{i}$.

Proposition 21 *RMA* admet un ratio compétitif de $2H_k$.

DÉMONSTRATION.
On considère une entrée $x_1 \dots x_n$ que l'on écrit comme $B_1 \dots B_t$, comme précédemment.
Les pages de B_i se divisent en deux groupes :

1. les nouvelles pages qui n'apparaissent pas dans B_{i-1} (ou pour $i = 0$, celles qui ne sont pas dans le cache initialement) ;
2. les anciennes pages qui apparaissent dans B_{i-1} (ou pour $i = 0$, celles qui sont déjà présentes dans le cache initialement).

Soit m_i le nombre de nouvelles pages dans B_i ; il y a $k - m_i$ anciennes pages dans B_i. La première page de B_i est nouvelle. A chaque traitement de la première page d'un bloc B_i, c'est un échec, et qui plus est, toutes les pages du cache sont marquées, et donc, toutes sont alors démarqués.



Fait 22 Chaque nouvelle page dans B_i va faire un échec (quand elle est demandée pour la première fois).

DÉMONSTRATION. Evident ! ■
Les anciennes pages, malheureusement, peuvent aussi faire un échec car certaines ont été remplacées par des nouvelles.

Fait 23 $\mathbb{P}$(échec sur la j-ème ancienne page) $\leq \frac{m_i}{k-j-1}$.

DÉMONSTRATION. Soit ℓ le nombre de nouvelles pages avant la j-ème. Voici l'état du cache au moment de l'examen de la j-ème ancienne page.
— les $j-1$-ères anciennes pages sont dans le cache et sont marquées (soit elles n'ont jamais été touchées, soit une nouvelle page est venu en supprimer une, puis elle a été rétablie ensuite)
— ℓ nouvelles pages sont présentes et marquées, et ont remplacées des pages du cache.
La j-ème ancienne page, si elle a été supprimée, c'est soit par une nouvelle page, soit par une ancienne, car l'ancienne a été supprimée par une nouvelle page. Dans tous les cas, c'est comme si elle a été supprimée par une nouvelle page. Donc la j-ème ancienne page ne doit pas faire partie des ℓ cases mémoires supprimées parmi les $k - (j-1)$ cases restantes (anciennes pages mais pas les $j-1$ anciennes pages déjà traitées).

$$\mathbb{P}(\text{échec sur la } j\text{-ème ancienne page}) = \mathbb{P}(\text{cette } j\text{-ème ancienne page ne soit pas dans le cache}) = \frac{\ell}{k-(j-1)}.$$

$\frac{m_i}{k-(j-1)}$. ■
Ainsi l'espérance du nombre d'échecs durant le traitement de B_i est majorée par

$$m_i + \frac{m_i}{k} + \frac{m_i}{k-1} + \frac{m_i}{k-2} \dots \frac{m_i}{k-(k-m_i-1)} \leq m_i(1 + \frac{1}{2} + \dots \frac{1}{k}) = m_i H_k$$

NB : comme $m_i \geq 1$, on a $k - (k - m_i - 1) \geq 2$. Ainsi *RMA* a une espérance d'échecs de $H_k \times \sum_{i=1}^{t} m_i$.

Fait 24 $OPT(x) \geq \frac{m_i}{2}$ échecs dans la phase B_i.

DÉMONSTRATION. Considérons un algorithme offline optimal. Soit F_i^* le nombre d'échecs dans la phase i.
— $F_1^* \geq m_i$;
— Le nombre de pages distinctes durant la phase $i-1$ et i est $k + m_i$. En effet, il y a k pages distinctes dans B_{i-1} et seulement m_i nouvelles pages dans B_i. Mais alors, l'algorithme offline, pour sûr fait au moins m_i fautes. Ainsi, $F_{i-1}^* + F_i^* \geq m_i$
Le nombre d'échecs total vaut :

$$F^* = \sum_{i=1}^{t} F_i^* \geq \frac{1}{2}(F_1^* + \sum_{i=2}^{t} F_{i-1}^* + F_i^*) \geq \frac{1}{2} \sum_{i=1}^{t} m_i.$$

■
■

4.5 Borne inférieure pour algorithme probabiliste

Théorème 25 Supposons qu'il y ait plus de $k+1$ pages, où k est la taille du cache. Tout algorithme probabiliste pour le problème de pagination a un facteur compétitif d'au moins H_k.

DÉMONSTRATION. Pour cela, nous allons définir une variable aléatoire sur les entrées X pour laquelle $c_A^X \leq H_k$ pour tout algorithme déterministe A.

La distribution pour X est : les $x_1, \ldots, x_n$ sont indépendants et on choisit x_i uniformément dans $\{1, \ldots, k+1\}$.

Fait 26 Pour tout algorithme déterministe ALG, $\mathbb{E}(ALG(X)) = \frac{n}{k+1}$.

DÉMONSTRATION. En effet, il y a une probabilité de $\frac{1}{k+1}$ de tomber sur une page non présente. On écrit :

$$ALG(X) = \sum_{j=1}^{n} Y_j$$

où $Y_j = 1$ si x_j fait un échec au moment où elle est traitée, 0 sinon.

En utilisant la linéarité de l'espérance, on a gagné :

$$\mathbb{E}(ALG(X)) = \sum_{j=1}^{n} \mathbb{E}(Y_j) = \frac{n}{k+1}.$$

■

Analysons maintenant la valeur de $OPT(X)$.

On divise X en paquet $B_1 \ldots B_{T(n)}$ comme précédemment, sauf que maintenant $T(n)$ est une variable aléatoire qui est le nombre de phases dans une entrée X de longueur n. On a $OPT(X) \leq T(n) + 1$. TODO: pourquoi ? Ainsi $\mathbb{E}(OPT(X)) \leq \mathbb{E}(T(n)) + 1$.

Par le *Elementary Renewal Theorem* : TODO: potasser ça un jour...

$$\lim_{n\to\infty} \frac{n}{\mathbb{E}(T(n))} = \mathbb{E}(W)$$

où W est comme B_1 si n étant infini... bref, il s'agit du *problème de collection de coupons.* Il s'agit d'acheter des boîtes de céréales, jusqu'à ce que nous ayons collectionner les $k+1$ types de jouets.

Fait 27 $\mathbb{E}(W) = (k+1)H_k$.

DÉMONSTRATION. On écrit $W := Z_1 + Z_2 + \ldots Z_k + Z_{k+1} - 1$ où Z_i = nb de pages vues avant de voir la i-ème nouvelle. Il faut faire -1 à la fin car sinon, on comptabilise la $k+1$-ème page, alors qu'elle est dans la phase suivante.

$Z_1 = 1$

La probabilité de voir une nouvelle page est $\frac{k}{k+1}$. Ainsi Z_2 est une variable de loi géométrique de paramètre $\frac{k}{k+1}$. Son espérance vaut $\mathbb{E}(Z_2) = \frac{k+1}{k}$.

De manière générale, Z_i est géométrique de paramètre $\frac{k-i+2}{k+1}$ et donc $\mathbb{E}(Z_i) = \frac{k+1}{k-i+2}$.

Par la linéarité de l'espérance, on a

$$\mathbb{E}(W) = 1 + (k+1)(\frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{2} + \frac{1}{1}) - 1 = (k+1)H_k.$$

■

On a

$$\begin{aligned}\lim_{n\to\infty} \frac{\mathbb{E}(ALG(X))}{\mathbb{E}(OPT(X))} &= \lim_{n\to\infty} \frac{n\mathbb{E}(ALG(X))}{n\mathbb{E}(OPT(X))} \\ &\geq \frac{(k+1)}{\mathbb{E}(|W|)} \\ &= H_k\end{aligned}$$

Démontrons l'inégalité du principe minmax de Yao. On a pour tout algorithme déterministe :

$$\mathbb{E}(ALG(X)) \geq H_k \mathbb{E}(OPT(X))$$

Considérons maintenant un algorithme probabiliste ALG_R où R est la variable aléatoire des bits aléatoires. On note alors $ALGR_r$ l'algorithmique déterministe qui suit les choix dictés par le mot aléatoire r. On a :

$$\mathbb{E}(ALG_r(X)) \geq H_k \mathbb{E}(OPT(X))$$

En passant à l'espérance sur r, i.e. on fait introduire la variable aléatoire R :

$$\mathbb{E}(ALG_R(X)) \geq H_k \mathbb{E}(OPT(X))$$

Mais alors il existe une instance $x = x_1, \ldots, x_n$ avec :

$$\mathbb{E}(ALG_R(x)) \geq H_k \mathbb{E}(OPT(x))$$

En effet, si ce n'était pas le cas, on aurait $\mathbb{E}(ALG_R(x)) < H_k \mathbb{E}(OPT(x))$ pour tout x, et donc $\mathbb{E}(ALG_R(X)) < H_k \mathbb{E}(OPT(X))$.

Ce qui montre que le ratio d'approximation est au moins de H_k.

■

5 Principe minmax de Yao

Théorème 28 (principe minmax de Yao, admis)

$$\min_{\text{ALGR}} \max_{x} \mathbb{E}(\text{coût de } ALGR(x)) = \max_{X} \min_{ALG} \mathbb{E}(\text{coût de } ALG(X))$$

où $ALGR$ dénote n'importe quel algorithme probabiliste, x une entrée, X une variable aléatoire sur les entrées, ALG n'importe quel algorithme déterministe.

Références bibliographiques

Les livres [FW98] et [BE98] (une nouvelle version est en train d'être écrite) sont les références sur le sujet. La section sur l'algorithme online pour les arbres de Steiner provient d'un cours de Tim Roughgarden.

Références

[BE98] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[FW98] Amos Fiat and Gerhard J. Woeginger, editors. *Online Algorithms, The State of the Art (the book grow out of a Dagstuhl Seminar, June 1996)*, volume 1442 of *Lecture Notes in Computer Science*. Springer, 1998.