Optimisation discrète sous incertitudes

Christian Artigues

LAAS-CNRS

2 octobre 2013

1. Optimisation en ligne
   - Algorithme en ligne
   - Analyse compétitive
   - Problème “Acheter ou louer”
   - Problème de pagination
   - Randomisation

Algorithme d'optimisation en ligne

- L'algorithme A doit servir séquentiellement un ensemble de *requêtes* $\sigma(1), \sigma(2), \ldots, \sigma(m)$
- "Servir" la requête i revient à prendre de manière définitive une décision sous la forme d'une *réponse* $\delta_i \in D_i$, ou D_i est l'ensemble des réponses possibles
- La séquence de décisions $(\delta_1, \delta_2, \ldots, \delta_m)$ donne la solution d'une instance [1] $I = (\sigma_i)_{i=1,\ldots,m}$ d'un problème d'optimisation progressivement révélée par les "reqûetes".
- L'ensemble des décisions prises a ainsi un coût $f^m(\delta_1, \delta_2, \ldots, \delta_m)$ qu'il s'agit de minimiser, $f^i(\delta_1, \delta_2, \ldots, \delta_i)$ donnant le coût intermédiaire des décisions $\delta_1, \ldots, \delta_i$.
- Mais chaque requête $\sigma(i)$ doit être servie (et donc la décision δ_i doit être prise) en ligne sans connaissance du futur, i.e. les requêtes $\sigma(i+1), \ldots, \sigma(m)$.

1. Instance : donnée d'un problème d'optimisation et de toutes les valeurs numériques des paramètres

Algorithme d'optimisation en ligne

Exemple

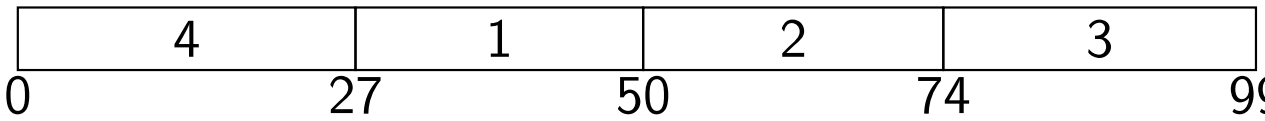
- Soit l'instance I d'un problème d'ordonnancement de 4 tâches de durées $p_1 = 23$, $p_2 = 24$, $p_3 = 25$, $p_4 = 27$ et de dates d'arrivées $r_1 = 2$, $r_2 = 2$, $r_3 = 1$, $r_4 = 0$. Les tâches ne peuvent pas être interrompues. Le coût est la somme des dates de fin des tâches.
- Hypothèses sur les incertitudes et les décisions :
  1. Les tâches ne sont connues qu'à leur date d'arrivée et leur durée est inconnue quand elles arrivent. La décision doit être prise à chaque date d'arrivée (FIFO).
  2. Les tâches ne sont connues qu'à leur date d'arrivée et leur durée est connue quand elles arrivent. La décision doit être prise à chaque date d'arrivée (FIFO).
  3. Les tâches ne sont connues qu'à leur date d'arrivée et leur durée est inconnue quand elles arrivent. La décision doit être prise à chaque date de libération de machine parmi les tâches arrivées.
  4. Les tâches ne sont connues qu'à leur date d'arrivée et leur durée est connue quand elles arrivent. La décision doit être prise à chaque date de libération de machine parmi les tâches arrivées.

Algorithme d'optimisation en ligne

Exemple (suite)

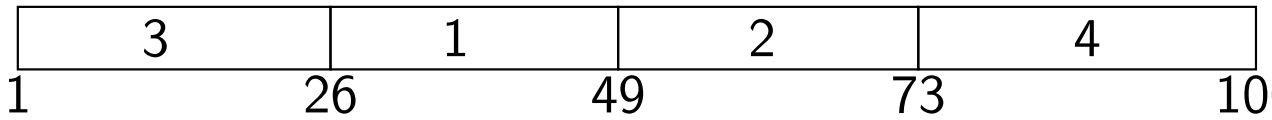
- Hypothèses 1 et 2, la séquence de requêtes est $\sigma(1) : t = 0$, ordonnancer la tâche 4, $\sigma(2) : t = 1$: ordonnancer la tâche 3, $\sigma(3) : t = 2$ ordonnancer la tâche 1 ou la tâche 2
- Hypothèses 3 et 4, la séquence de requêtes est $\sigma(1) : t = 0$, ordonnancer la tâche 4, $\sigma(2)$, $t = 27$: ordonnancer la tâche 1 ou la tâche 2 ou la tâche 3.
- Si on connait la durée des tâches lorsqu'elles arrivent (hypothèses 2 et 4) l'algorithme peut exploiter cette information (par exemple plus petite durée d'abord). Sinon (hypothèses 1 et 3), ordonnancer les tâches au hasard ?

Ordonnancement en ligne (hypothèse 4+SPT)



$\sum C_i = 250$

Ordonnancement optimal :



$\sum C_i = 248$

Analyse compétitive

- Soit $f^*(I) = \min\{f^m(\delta) | \delta \in D_1 \times D_2 \times \ldots \times D_m\}$ la valeur optimale compte tenu des décisions possibles à chaque étape.
- Soit $f(I, A) = f^m(\delta^A)$ la valeur obtenue par l'algorithme A avec les décisions $\delta_1^A, \ldots, \delta_m^A$.
- **Facteur de compétitivité** sur une instance I (minimisation) :

$$\rho(A, I) = f(I, A)/f^*(I)$$

  Exemple pour l'instance précédente $\rho(A, I) \simeq 1,02$.
- **Facteur de compétitivité dans le pire des cas** : pire facteur de compétitivité sur l'ensemble des instances $\mathcal{I}$

$$\rho(A) = \max_{I \in \mathcal{I}} f(I, A)/f^*(I)$$

- **Algorithme c-compétitif** : Un algorithme A est c-compétitif si $\rho(A) \leq c$

Objectif : obtenir un algorithme en ligne du meilleur facteur de compétitivité possible.

Analyse compétitive

- Soit $f^*(I) = \min\{f^m(\delta) | \delta \in D_1 \times D_2 \times \ldots \times D_m\}$ la valeur optimale compte tenu des décisions possibles à chaque étape.
- Soit $f(I, A) = f^m(\delta^A)$ la valeur obtenue par l'algorithme A avec les décisions $\delta_1^A, \ldots, \delta_m^A$.
- **Facteur de compétitivité** sur une instance I (minimisation) :

$$\rho(A, I) = f(I, A)/f^*(I)$$

  Exemple pour l'instance précédente $\rho(A, I) \simeq 1,02$! Problème !.
- **Facteur de compétitivité dans le pire des cas** : pire facteur de compétitivité sur l'ensemble des instances $\mathcal{I}$

$$\rho(A) = \max_{I \in \mathcal{I}} f(I, A)/f^*(I)$$

- **Algorithme c-compétitif** : Un algorithme A est c-compétitif si $\rho(A) \leq c$

Objectif : obtenir un algorithme en ligne du meilleur facteur de compétitivité possible.

Analyse compétitive

Solution de l'exercice "Acheter ou louer"

On considère un coût de B pour acheter et un côut de 1 pour louer. Une requête correspond aux décisions : "acheter ou louer". L'inconnue est limitée ici aux nombres de requêtes m (nombre de jours de ski).

Si $m \geq B$, l'optimum est d'acheter à la première requête (coût B). Si $m \leq B$ l'optimum est de toujours louer (coût m).

Pour évaluer le facteur de compétitivité d'une décision, on introduit le concept d'adversaire. si j'achète à la requête k à combien l'adversaire va fixer m pour me faire perdre le plus d'argent possible ?

Si $1 \leq k \leq B$, le coût est de $(k-1)+B$ et le pire des cas est $m = k$ avec un optimum de k ce qui donne un facteur de compétitivité de $1 + \frac{B-1}{k}$. si $k \geq B$, le coût est aussi de $(k-1)+B$ et le pire des cas est aussi $m = k$ avec un optimum de B et un facteur de compétitivité de $1 + \frac{k-1}{B}$.

L'algorithme optimal est de choisir la valeur de k qui minimise ces fonctions : $k = B$ avec un facteur de compétitivité de $2 - \frac{1}{B}$. Pour $B = 10$ on obtient $1,9$.

On considère un système informatique ou un utilisateur demande l'accès à des pages stockées en mémoire. Il y a en tout n pages et l'utilisateur demande une séquence (inconnue du système) des n pages.

Les n pages sont stockées sur une mémoire de taille suffisante mais d'accès lent. On dispose également d'une mémoire d'accès rapide mais pouvant stocker un nombre limité $k \ll n$ de pages.

Etant donné une séquence $\sigma(1), \sigma(2), \ldots, \sigma(m)$ de pages demandées à partir d'une mémoire rapide pleine, on doit concevoir un algorithme en ligne qui, si la prochaine page $\sigma(i)$ n'est pas en mémoire la charge dans la mémoire rapide en remplaçant une des k pages présentes.

Mémoire rapide ($k=5$) : | 7 | 2 | 5 | 1 | 9 | Séq. : 3,9,4,3,2,1,2,7...
La politique optimale consiste à remplacer la page qui sera demandée le plus tard (ici la page 5 qui n'est pas demandée). On obtient | 7 | 2 | 3 | 1 | 9 |

On considère un système informatique ou un utilisateur demande l'accès à des pages stockées en mémoire. Il y a en tout n pages et l'utilisateur demande une séquence (inconnue du système) des n pages.

Les n pages sont stockées sur une mémoire de taille suffisante mais d'accès lent. On dispose également d'une mémoire d'accès rapide mais pouvant stocker un nombre limité $k \ll n$ de pages.

Etant donné une séquence $\sigma(1), \sigma(2), \ldots, \sigma(m)$ de pages demandées à partir d'une mémoire rapide pleine, on doit concevoir un algorithme en ligne qui, si la prochaine page $\sigma(i)$ n'est pas en mémoire la charge dans la mémoire rapide en remplaçant une des k pages présentes.

Mémoire rapide ($k=5$) : | 7 | 2 | 5 | 1 | 9 | Séq. : 3,9,4,3,2,1,2,7...
La politique optimale consiste à remplacer la page qui sera demandée le plus tard (ici la page 5 qui n'est pas demandée). On obtient | 7 | 2 | 3 | 1 | 9 |

Problème : pas implémentable si la séquence est inconnue !

Exemple d'algorithmes de pagination :

- LRU (Least Recently Used). Remplacer la page qui a été demandée le moins récemment.
- FIFO (First-In First-Out). Remplacer la page qui est dans la mémoire depuis le plus de temps.

Théorème

Si la taille de la mémoire rapide est k, LRU et FIFO sont k-compétitifs.

Preuve au tableau □

Théorème

Si un algorithme déterministe A est $c-$compétitif alors $c \geq k$.

Preuve au tableau □

- Un algorithme randomisé RA peut être vu comme un ensemble d'algorithmes déterministes $\{A_y\}$ associé à une distribution de probabilité Y.

Exemple

Pour le problème P d'"Acheter ou louer" considérons l'algorithme non déterministe $\tilde{A}_B$ suivant muni d'une distribution de probabilité discrète. Soit A_B l'algorithme déterministe consistant à acheter à la Bème journée de ski :

$$\tilde{A}_B(p) = \begin{cases} A_{B/2} & \text{avec la probabilité } p \\ A_B & \text{avec la probabilité } (1-p) \end{cases}$$

Randomisation

Types d'adversaires

Lorsque l'algorithme RA peut choisir des décisions au hasard, les concepts de la pire instance générée par un adversaire et du coût optimal sur l'instance (le coût payé par l'adversaire) peuvent être déclinés selon trois possibilités :

- **Adversaire faible** (*Oblivious adversary*): l'adversaire génère l'instance I en connaissant l'algorithme RA, mais sans connaître les résultats d'aucune des décisions aléatoires (l'instance doit être générée à l'avance). L'adversaire paye le coût optimal $f^*(I)$.
- **Adversaire moyen** (*Adaptative online adversary*) : l'adversaire génère l'instance I_{RA} en adaptant les requêtes en fonction des décisions déjà prises, mais doit prendre sa décision **en ligne** sans connaitre la décision présente et future de RA. En fait l'adversaire se comporte comme un algorithme déterministe en ligne $\hat{A}$ mais il peut générer les requêtes pour payer le moins cher. Il paye au final $f(I_{RA}, \hat{A})$.
- **Adversaire fort** (*Adaptative offline adversary*) : l'adversaire génère l'instance I_{RA} de manière adaptative (comme ci-dessus) mais paye le coût optimal de l'instance $f^*(I_{RA})$.

- L'adversaire faible correspond à la transposition dans le cas non déterministe du facteur de compétitivité.
- Un algorithme randomisé RA est c-compétitif contre tout adversaire faible si, pour toute instance $I = (\sigma_i)_{i=1,\ldots,m}$,

$$\mathbb{E}_Y[f(I, RA_y)] \leq cf^*(I)$$

- Un algorithme randomisé RA est c-compétitif contre un adversaire moyen (associé à un algorithme en ligne $\hat{A}$) si

$$\mathbb{E}_Y[f(I, RA_y) - cf(I_{RA}, \hat{A})] \leq 0$$

- Un algorithme randomisé RA est c-compétitif contre un adversaire fort si

$$\mathbb{E}_Y[f(I, RA_y) - cf^*(I_{RA})] \leq 0$$

où I_{RA} est l'instance générée de manière adaptative par l'adversaire.

Pour un problème d'optimisation envligne donné (ensemble des requêtes et des réponses possibles et fonction de coût associée)

- **Théorème**. S'il existe un algorime randomisé c-compétitif contre n'importe quel adversaire fort alors il existe aussi un algorithme déterministe $c-$compétitif.
- **Théorème**. Si RA est un algorithme randomisé c-compétitif contre n'importe quel adversaire moyen et s'il existe un algorithme randomisé d-compétitif contre n'importe quel adversaire faible, alors RA est $c.d$-compétitif contre n'importe quel adversaire fort.
- **Corollaire**. Si RA est un algorithme randomisé c-compétitif contre n'importe quel adversaire moyen, alors il existe un algorithme déterministe c^2-compétitif.

Exercice

Pour le problème P d'"Acheter ou louer" considérons l'algorithme non déterministe $\tilde{A}_B$ suivant muni d'une distribution de probabilité discrète. Soit A_B l'algorithme déterministe consistant à acheter à la Bème journée de ski :

$$\tilde{A}_B(p) = \begin{cases} A_{B/2} & \text{avec la probabilité } p \\ A_B & \text{avec la probabilité } (1-p) \end{cases}$$

Calculer la probabilité p qui minimise le facteur de compétitivité $\max_{j \in \mathbb{N}} \mathbb{E}[f(j, \tilde{A}_B)(p)]/f^*(j)$ contre tout adversaire faible où j est le nombre de jours de ski (définissant une instance).

Calcul au tableau...

On obtient $p = \frac{2}{5}\left(1 - \frac{1}{5B-4}\right)$ et $\max_I \mathbb{E}[f(I, \tilde{A}_B(p))] = \frac{9}{5} - \frac{1}{B}\left(1 - \frac{B}{25B-20}\right)$, ce qui pour $B = 10$ donne $p \approx 0,391$ et un facteur de compétitivité en espérance d'environ 1,704.

On considère l'algorithme de marquage suivant:

- On part d'un ensemble de k pages *non marquées*. A chaque requête, la page demandée est marquée. En cas d'échec, on sélectionne aléatoirement (selon une loi uniforme) une page non marquée pour l'évincer. Lorsque toutes les pages en mémoire rapide sont marquées, et qu'un échec survient, on *démarque* toutes les pages et on réitère le processus.

Théorème

L'algorithme de marquage est $2H_k$-compétitif contre n'importe quel adversaire faible où $H_k = \sum_{i=1}^{k} 1/i$ est le k-ième nombre harmonique.

Démonstration au tableau. □

Remarque : $H_k \underset{+\infty}{\sim} ln(k)$. L'algorithme randomisé est donc exponentiellement plus compétitif que l'algorithme déterministe !

Randomisation

Principe du minimax de Yao

Etant donné un problème d'optimisation en ligne, le facteur de compétitivité du meilleur algorithme en ligne randomisé contre un adversaire faible est égal au facteur de compétitivité du meilleur algorithme en ligne déterministe avec la pire distribution de probabilité de la séquence de requêtes d'entrée. Plus formellement :

Théorème

- Soit $\rho(R)$ le facteur de compétitivité d'un algorithme randomisé $R \in \mathcal{R}$ contre tout adversaire faible avec $\mathcal{R}$ ensemble des algorithmes en ligne randomisés pour le problème.
- Soit $P \in \mathcal{P}$ une distribution de probabilités pour sélectionner la séquence de requêtes, avec $\mathcal{P}$ l'ensemble des distributions de probabilités.
- Soit c_A^P l'espérance du facteur de compétitivité d'un algorithme déterministe $A \in \mathcal{A}$ si les requêtes sont générées selon P, avec $\mathcal{A}$ l'ensemble des algorithmes en ligne déterministes pour le problème.

$$\min_{R \in \mathcal{R}} \rho(R) = \max_{P \in \mathcal{P}} \min_{A \in \mathcal{A}} c_A^P$$

On en déduit une technique pour borner inférieurement le facteur de compétitivité d'un algorithme en ligne contre un adversaire faible. Il suffit de choisir une distribution de probabilités particulière P et de trouver une borne inférieure LB telle que $\min_{A \in \mathcal{A}} c_A^P \geq LB$. On a alors $\min_{R \in \mathcal{R}} \rho(R) \geq LB$.

Considérons le problème du skieur avec un coût $B = 10$. Soit la distribution de probabilité P telle qu'il n'y ait qu'un seul jour de ski avec une probabilité $1/2$ et 20 jours de ski avec une probabilité $1/2$.

L'espérance du côut optimal est $\mathbb{E}[c_{OPT}] = \frac{1}{2}.1 + \frac{1}{2}.10 = 11/2$.

Soit A_j l'algorithme déterministe qui achète le jour j. On a

$$\mathbb{E}[\rho(A_j)] = \begin{cases} 10 & \text{pour } j = 1 \\ \frac{1}{2}.1 + \frac{1}{2}(j - 1 + 10) & \text{pour } j = 2, \ldots, 20 \\ \frac{1}{2}.1 + \frac{1}{2}.20 & \text{pour } j > 20 \end{cases} \quad \geq \frac{1}{2} + \frac{11}{2} = 6.$$

On a alors $c_{A_j}^P = \frac{\mathbb{E}[\rho(A_j)]}{\mathbb{E}[c_{OPT}]} \geq \frac{12}{11}$. Aucun algorithme en ligne ne peut être c-compétitif avec $c \geq 12/11$.

sources

- **Susanne Albers**. BRICS. Mini course on Competitive Online Algorithms. MPI Saarbrücken. **Aarhus University**. August 27–29 1996. http://www.brics.dk/LS/96/2/
- **Sven O. Krumke**. Online Optimization. OptALI Summer School Auckland, 2011. **Technische Universität KaiserSlautern**.
- **Jim Sukha**. 6.046, Introduction to Algorithms. Recitation Notes on Competitive analysis. **MIT** http://people.csail.mit.edu/sukhaj/6.046/rec9.pdf
- **Luca Trevisan**. CS261 Lecture 17: Online Algorithms. **Standford University**. http://lucatrevisan.wordpress.com/2011/03/09/cs261-lecture-17-online-algorithms/
- **Rudolf Fleischer**, COMP 670K- Topics in Theoretical CS - Online Algorithms - Fall 2000, **Fudan University** http://www.tcs.fudan.edu.cn/rudolf/Courses/Online/Online00/index.html