

# Cours 8: Algorithmes online

- Offline / Online, compétitivité
- Bin packing, lien avec algo d'approx
- Cache paging, adversaire, borne inférieure
- Accès de liste, méthode du potentiel
- Les  $k$  serveurs, adversaires multiples
- Robots et exploration

# Online contre Offline Vélib ou Gitane?

**Offline:** toutes les données sont disponibles dès le début

**Online:** il faut décider au fur et à mesure de l'arrivée des données

**Compétitivité:** Un algorithme online est  $\alpha$ -compétitif si  
pour toute entrée  $x$ ,  $\text{online}(x) \leq \alpha \cdot \text{OPT}(x) + cte$

**L'exemple de base:**

Louer un vélib pour une journée coûte 1 euro. Acheter un vélo coûte 250 euros. Comment ne pas trop dépenser, sachant qu'un jour ou l'autre on arrêtera le vélo...

**Donnée:** La suite des jours: V,V,V,V,F.

**Algorithme on-line:** louer le vélo 250 fois puis l'acheter.

Cet algorithme est 2-compétitif: l'algorithme optimal décide en fonction du nombre  $J$  de jours d'utilisation du vélo

- Si  $J < 250$  alors  $\text{online}=\text{OPT}$
- Si  $J \geq 250$  alors  $\text{online}=500$  et  $\text{OPT}=250$ .

# Portefeuille boursier, algo probabiliste

**Problème:** vendre un portefeuille d'actions au cours d'une période fixe, sachant que les prix fluctureront entre un prix plancher  $p$  et un prix plafond  $P$

**Algo palier (déterministe):** fixer un prix palier  $q$  et vendre dès que ce prix est atteint, sinon vendre au prix final, ie au moins  $p$ .

**Analyse:**

- si le prix max  $m$  observé est  $< q$ , au pire on a vendu à  $p$ : ratio  $\leq q/p$ ;
- sinon  $m \geq q$  et on a vendu à  $q$ : ratio  $\leq P/q$ .

Le pire cas  $\max(q/p, P/q)$  est minimum pour le choix  $q = \sqrt{pP}$

$\Rightarrow$  l'algo est  $\sqrt{P/p}$ -compétitif.

**Algo palier par fraction (déterministe):** supposons  $P = p2^k$ ,  
vendre  $\frac{1}{k}$  des actions quand on atteint  $p2^i$  pour  $i = 0, \dots, k - 1$

**Analyse:** si  $m$  est entre  $p2^j$  et  $p2^{j+1}$ , on a gagné  $\frac{1}{k} \cdot p(1 + 2 + \dots + 2^j)$   
contre  $m \leq p2^{j+1}$  soit un ratio de  $k = \log_2 P/p$ .

$\Rightarrow$  l'algo est  $(\log_2 P/p)$ -compétitif en moyenne 

**Algo probabiliste:** choisir  $i \in [0, k[$  avec proba  $\frac{1}{k}$  puis algo palier à  $q = p2^i$ .

# Cours 8: Algorithmes online

- Offline / Online, compétitivité
- Bin packing, lien avec algo d'approx
- Cache paging, adversaire, borne inférieure
- Accès de liste, méthode du potentiel
- Les  $k$  serveurs, adversaires multiples
- Robots et exploration

# Rangement optimal et approximation gloutonne

**Donnée:**  $n$  objets de poids  $p_1, \dots, p_n$ , et des boites de capacité  $P$ .

**Problème:** Mettre les objets dans un nombre minimum de boites.

On a vu un algorithme glouton approché à un facteur  $3/2$ , qui classe les objets par poids décroissant et les range de manière gloutonne.

**Version Online:** on ne peut plus classer les objets par poids décroissant, on doit les traiter au fur et à mesure qu'ils arrivent.

**Algorithme glouton Online:** on place les objets dans l'ordre où ils arrivent en n'ouvrant une nouvelle boite que si c'est nécessaire.

**Analyse:** similaire à l'analyse de l'algo précédent.

il faut au moins une nouvelle boite pour chaque objet de taille  $> P/2$   
si l'algo online place un gros objet dans chaque boite, il est optimal  
sinon considérons la dernière boite ouverte qui n'ait pas de gros objet  
l'objet pour lequel elle a été ouverte n'entraîne dans aucune des autres boites

$\Rightarrow$  toutes les boites sauf une sont au moins  $\frac{1}{2}$ -pleine

$$\text{Glouton Online} \leq 2 \cdot \text{OPT} + 1$$

# Cours 8: Algorithmes online

- Offline / Online, compétitivité
- Bin packing, lien avec algo d'approx
- Cache paging, adversaire, borne inférieure
- Accès de liste, méthode du potentiel
- Les  $k$  serveurs, adversaires multiples
- Robots et exploration

# Gestion du cache mémoire, borne inférieure

**Donnée:** mémoire centrale organisée en pages, mémoire cache de  $k$  pages; le chargement d'une page coûte 1 (et on sort une des pages du cache).

**Problème:** être  $\alpha$ -compétitif pour toute séquence de consultations de pages. Une politique de cache consiste à choisir la page qui est éliminée du cache lorsqu'une nouvelle page doit entrer, exemple: LRU (least recently accessed), FIFO (first in first out), LFU (least frequently used).

Comparer à OPT (offline): éliminer la page qui sera redemandée le plus tard

**Analyse:** pour tout algorithme online  $k$ -page déterministe on peut construire une séquence sur  $k + 1$  pages qui nécessite de faire entrer une nouvelle page à chaque consultation (le faire en se plaçant en *adversaire* de l'algorithme)

L'algo optimal offline pour cette séquence élimine la page parmi celles présentes qui sera redemandée le plus tard, *i.e.* dans  $k$  tours au moins, il n'aura donc pas besoin de faire entrer d'autre page avant  $k$  tours.

⇒ un algo online déterministe est au mieux  $k$ -compétitif

# Gestion du cache mémoire, algorithme déterministe

un algo online déterministe est au mieux  $k$ -compétitif

peut il vraiment être pire ?

Oui: par exemple Least Frequently Used n'est pas  $k$ -compétitif (cf PC)

Il n'est en fait pas compétitif, *i.e.* il n'est  $\alpha$ -compétitif pour aucun  $\alpha$ .

**Théorème.** LRU et FIFO sont  $k$ -compétitifs.

LRU et FIFO: dans les deux cas, si on paye  $k$  alors OPT doit payer au moins 1, car cela veut dire que  $k$  pages différentes sont entrées.

En PC: on peut faire mieux avec un algorithme probabiliste

# Cours 8: Algorithmes online

- Offline / Online, compétitivité
- Bin packing, lien avec algo d'approx
- Cache paging, adversaire, borne inférieure
- Accès de liste, méthode du potentiel
- Les  $k$  serveurs, adversaires multiples
- Robots et exploration

# Accès séquentiels dans une liste

**Données:** Une liste simplement chaînée  $L$ .

**Problème:** coût min pour traiter une suite d'accès (on peut réorganiser  $L$ )

L'accès à la clef en position  $i$  coûte  $i$  opérations (accès séquentiel). Un échange de clef voisines coûte 1 et on peut gratuitement amener la clef à laquelle on accède en tête de liste.

**Théorème (admis):** la recherche de la séquence d'opérations optimale offline est NP-dure.

**Exemple d'algo online:**

- Transpose: la clef demandée est avancée d'une place
- Compteur: les clefs sont classées par nb d'accès décroissants.
- Move-to-front: la clef demandée est mise en tête de liste

**Résultat:** Transpose et Compteur non compétitif, Move-to-Front l'est.

Pour Transpose et compteur il suffit d'exhiber des requêtes pénibles (cf PC)

Pour Move-to-front il faut un moyen de comparer à OPT...

# Méthode du potentiel

La méthode du potentiel peut être utile pour déterminer le facteur de compétitivité d'un algo  $A$  lorsqu'on ne connaît pas l'OPT.

L'idée est d'associer un potentiel  $\phi$  mesurant la différence entre les configurations dans lesquelles se trouvent  $A$  et OPT

On note  $\phi_i$  le potentiel après l'étape  $i$ ,  $A_i$  le coût de l'étape  $i$  pour l'algo  $A$ .

Le coût amorti d'une étape est  $a_i = A_i + \phi_i - \phi_{i-1}$ : on espère ainsi compenser une étape coûteuse par une baisse de potentiel.

**Proposition:** si le coût amorti  $a_i$  est inférieur à  $c \cdot OPT_i$ , et si le potentiel final est positif, alors  $A$  est  $c$ -compétitif.

$$A = \sum_i A_i = \phi_0 - \phi_k + \sum_i a_i \leq \phi_0 + \sum_i c \cdot OPT_i = \phi_0 + c \cdot OPT$$

Cette méthode est inspirée de l'analyse amortie des structures de données

# Accès dans une liste: Move-to-front

**Théorème:** Move-to-front est 2-compétitif

On considère la fonction potentiel suivante:

$\phi$  = nombre d'inversions entre la liste de MTF et celle d'OPT.

Étudions le coût du traitement de la  $i$ ème requête:

$OPT_i = k + \ell + 1 + E_i$  et  $A_i = k + m + 1$ , où  $E_i$  est le nb d'échanges effectués par OPT,  $k$  est le nb d'éléments devant  $r_i$  dans les 2 listes,  $\ell$  et  $m$  le nb de ceux qui sont devant dans l'une et pas dans l'autre

mais on a aussi:  $\phi_i - \phi_{i-1} \leq k - m + E_i$ : le nb d'inversions change en fonction des opérations effectuées.

Finalement le coût amorti  $a_i = A_i + \phi_i - \phi_{i-1} \leq 2 \cdot OPT_i$

Mais  $\sum A_i = \phi_0 - \phi_k + \sum a_i$  et donc  $A \leq cte + 2 \cdot OPT$ .

( $\phi_0$  ne dépend que de la longueur de la liste  $L$ )

# Cours 8: Algorithmes online

- Offline / Online, compétitivité
- Bin packing, lien avec algo d'approx
- Cache paging, adversaire, borne inférieure
- Accès de liste, méthode du potentiel
- Les  $k$  serveurs, adversaires multiples
- Robots et exploration

# Le problème des $k$ serveurs

**Donnée:** un espace métrique (inégalité triangulaire), le nb  $k$  de serveurs

**Requête:** une suite de points  $v_1 v_2 \dots$

**Problème:** pour chaque  $v_i$  déplacer un (ou plusieurs serveurs) de façon à en mettre un en  $v_i$ , en minimisant la somme des déplacements.

Modélise la gestion d'un service d'intervention : pas de retour à la base entre les interventions.

Espace continu ou ensemble fini de points (matrice des distances)

Si tous les points sont à égale distance  $\Leftrightarrow$  politique de cache

# Borne inf par adversaires multiples

**Donnée:** un espace métrique (inégalité triangulaire), le nb  $k$  de serveurs

**Requête:** une suite de points  $v_1 v_2 \dots$

**Problème:** pour chaque  $v_i$  déplacer un (ou plusieurs serveurs) de façon à en mettre un en  $v_i$ , en minimisant la somme des déplacements.

**Théorème:** un algo online est au mieux  $k$ -compétitif

On considère le cas de  $k$  serveurs pour  $k + 1$  points et un algorithme  $A$  qui démarre avec ses serveurs en  $x_1, \dots, x_k$ , et qui doit répondre à la requête calamiteuse lui demandant toujours le point vide.

Définissons  $k$  autres algorithmes  $A_i$ , le  $i$ ème initialisé avec  $x_i$  vide.

Lorsque  $A$  déplace un serveur de  $x_i$  à  $x_j$ , l'unique autre algorithme ayant  $x_i$  libre déplace son serveur de  $x_j$  en  $x_i$ .

Les points libres des  $k + 1$  algos sont ainsi toujours distincts.

On a alors  $A(v) = \sum_{i=1}^k A_i(v) \geq k \cdot \min_i A_i(v)$

# Le problème des $k$ serveurs: Algo Double-Service

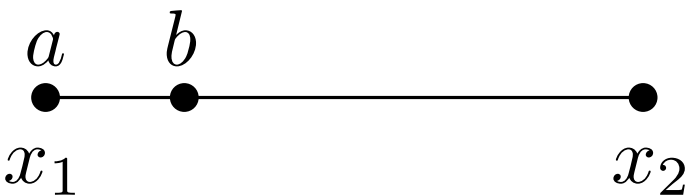
Considérons le cas où l'espace est une ligne:

Les  $k$  serveurs sont aux points  $x_1 < \dots < x_k$ , la requête est un point  $x$ .

Si  $x < x_1$  ou  $x > x_k$  on bouge  $x_1 := x$  ou  $x_k := x$  respectivement.

Plus généralement on déplace le serveur le plus proche ?

Non: si la situation est :



$x_1$   $x_2$

la requête *bababababa...* a un coût linéaire alors qu'il suffirait de mettre  $x_2$  en  $b$  pour avoir un coût constant.

**Algo double service:** si  $x$  est entre  $x_i$  et  $x_{i+1}$  approcher  $x_i$  et  $x_{i+1}$  à même vitesse jusqu'à ce que l'un des deux atteigne  $x$

**Théorème:** Double service est  $k$ -compétitif.

En PC: L'algo peut s'adapter aux arbres.

# Le problème des $k$ serveurs: service double

**Algo double service:** si  $x$  est entre  $x_i$  et  $x_{i+1}$  approcher  $x_i$  et  $x_{i+1}$  à même vitesse jusqu'à ce que l'un des deux atteigne  $x$

**Théorème:** Double service est  $k$ -compétitif.

On utilise la méthode du potentiel, avec un  $\phi \geq 0$  tel que les déplacements de coût  $x$  de OPT induisent un  $\Delta\phi \leq kx$  et ceux de  $A$  un  $\Delta\phi \leq -x$ .

En effet on aura alors  $\phi_n - \phi_0 \leq k \cdot OPT - A$  et donc  $A \leq k \cdot OPT + cte$

Le  $\phi$  utilisé est  $\phi = k\psi + \vartheta$  où  $\psi$  est le poids du couplage minimal entre les serveurs de OPT et ceux de  $A$  (pondéré par leur distance) et  $\vartheta$  est la somme des distances 2 à 2 des serveurs de  $A$ .

Il reste à vérifier que les déplacements de serveur induisent des variations de  $\phi$  comme indiquée ci-dessus.

# Cours 8: Algorithmes online

- Offline / Online, compétitivité
- Bin packing, lien avec algo d'approx
- Cache paging, adversaire, borne inférieure
- Accès de liste, méthode du potentiel
- Les  $k$  serveurs, adversaires multiples
- Robots et exploration



# Des robots aveugles: borne inf ?

**Théorème:** Tout algo est au mieux  $c\sqrt{n}$ -compétitif.

On construit pour tout robot  $A$  un ensemble de  $n/2$  rectangles  $1 \times n$  tel que le chemin parcouru par le robot est de longueur au moins  $n^2$  alors qu'il existe une ligne droite qui coupe au plus  $\sqrt{n}$  rectangle, et donc un chemin de longueur au plus  $n^{3/2}$ .

- Soit  $y$  l'abscisse à laquelle le robot arrive pour la 1ère fois en colonne  $2i + 1$ . On y place un rectangle  $1 \times n$  centré à hauteur  $y$ . Le robot doit alors parcourir une distance  $n/2$  au moins avant de pouvoir aller sur la droite.
- Dans une zone de hauteur  $n^{3/2}$  on peut placer  $\sqrt{n}$  lignes horizontales espacées d'une hauteur  $n/2$ . Un rectangle coupe au plus 2 lignes donc il existe une ligne qui en coupe moins de  $\sqrt{n}$ .
- Il faut monter au plus de  $n^{3/2}$  pour trouver la ligne, puis faire  $\sqrt{n}$  contournement de longueur  $n$ .

# Des robots aveugles: algo $\sqrt{n}$ -compétitif

**Théorème:** Il existe un algo  $c\sqrt{n}$ -compétitif.

Il faut trouver une façon de se déplacer qui garantisse un coût minimal pour le chemin optimal: on utilise des phases de balayage de bande horizontale...

Definition d'une phase de paramètre  $\tau$  et largeur  $L$ .

Coût de la traversée optimale d'une bande balayée dans les 2 directions.

Balayages successifs et bornes sur le nombre de balayage nécessaire en fonction de la longueur du chemin optimal.

Découverte de la longueur du chemin optimal par exponentiation.

# Cours 8: Algorithmes online

- Offline / Online, compétitivité
- Bin packing, lien avec algo d'approx
- Cache paging, adversaire, borne inférieure
- Accès de liste, méthode du potentiel
- Les  $k$  serveurs, adversaires multiples
- Robots et exploration

**Retenir:** Online/Offline,  $\alpha$ -compétitivité

Liens avec algos approchés, notions d'adversaire, de potentiel.

**En savoir plus:** INF581; cours d'Adi Rosén au MPRI.