

Cours 9: Algorithmes online

- Offline / Online, compétitivité
- Velib ou Gitane, l'exemple historique
- Bin packing, lien avec algo d'approx
- Portefeuille boursier, algo probabiliste
- Paging, borne inférieure

Online contre Offline

Offline: toutes les données sont disponibles dès le début

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

Online contre Offline

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 α -compétitif si
pour toute entrée x , $\text{online}(x) \leq \alpha \cdot \text{optimal-offline}(x)$

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 α -compétitif si
pour toute entrée x , $\text{online}(x) \leq \alpha \cdot \text{optimal-offline}(x)$

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.

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 α -compétitif si
pour toute entrée x , $\text{online}(x) \leq \alpha \cdot \text{optimal-offline}(x)$

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.

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 α -compétitif si
pour toute entrée x , $\text{online}(x) \leq \alpha \cdot \text{optimal-offline}(x)$

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

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 α -compétitif si
pour toute entrée x , $\text{online}(x) \leq \alpha \cdot \text{optimal-offline}(x)$

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 $\text{online}=\text{optimal}$

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 α -compétitif si
pour toute entrée x , $\text{online}(x) \leq \alpha \cdot \text{optimal-offline}(x)$

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

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.

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.

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.

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.

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 $> P/2$

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

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

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

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

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

$$\text{Online} \leq 2 \cdot \text{Opt} + 1$$

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

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: fixer un prix palier q et vendre dès que ce prix est atteint, sinon vendre au prix final, ie au moins p .

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: 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 $\leq P/q$.

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: 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 $q = \sqrt{pP}$

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

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: 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 $q = \sqrt{pP}$

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

Algo palier par fraction: supposons $P = p2^k$,
vendre $\frac{1}{k}$ des actions quand on atteint $2^i \cdot m$.

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: 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 $q = \sqrt{pP}$

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

Algo palier par fraction: supposons $P = p2^k$,
vendre $\frac{1}{k}$ des actions quand on atteint $2^i \cdot m$.

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 $2k = 2 \log_2 P/p$.

\Rightarrow l'algo est $(2 \log_2 P/p)$ -compétitif

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: 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 $q = \sqrt{pP}$

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

Algo palier par fraction: supposons $P = p2^k$,
vendre $\frac{1}{k}$ des actions quand on atteint $2^i \cdot m$.

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 $2k = 2 \log_2 P/p$.

\Rightarrow l'algo est $(2 \log_2 P/p)$ -compétitif

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

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: 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 $q = \sqrt{pP}$

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

Algo palier par fraction: supposons $P = p2^k$,
vendre $\frac{1}{k}$ des actions quand on atteint $2^i \cdot m$.

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 $2k = 2 \log_2 P/p$.

\Rightarrow l'algo est $(2 \log_2 P/p)$ -compétitif en moyenne 

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

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 α -compétitif pour toute séquence de consultations de pages.

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 α -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), MFU (most frequently used).

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 α -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), MFU (most frequently used).

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

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 α -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), MFU (most 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)

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 α -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), MFU (most 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.

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 α -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), MFU (most 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, 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 α -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), MFU (most 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

LRU, FIFO et tout algo de coût inférieur au nb de pages \neq dans la requête.

Cours 9: Algorithmes online

- Offline / Online, compétitivité
- Velib ou Gitane, l'exemple historique
- Bin packing, lien avec algo d'approx
- Portefeuille boursier, algo probabiliste
- Paging, borne inférieure

Retenir: Online/Offline, α -compétitivité

Liens avec algos approchés, probabilistes, notion d'adversaire.