Algorithmes

Algorithmes gloutons

CM 9 • Le problème de l'allocation d'une ressource • 16 novembre 2021

F. Laroussinie
M1 – Algo
2021 – 2022

Principes

Algorithmes gloutons *(greedy algo)*

A chaque étape de l'algo, **on fait un choix « localement optimal »**, puis on se ramène à la résolution d'un sous-problème.

↔ Facile à faire...

⇒ **Top down + efficaces**

- Dans cette famille d'algorithmes, on est capable de faire un bon choix à chaque étape de l'algorithme qui nous permet de passer à la résolution d'un sous-problème : le choix **« localement optimal »** conduit à une solution optimale (le choix est donc aussi « globalement optimal »).
- Notons que la notion d'optimalité renvoie à une vision « problèmes d'optimisation » qui aide à voir la structure de ces algorithmes mais ces algorithmes ne s'appliquent pas seulement à des problèmes d'optimisation.
- Ce type d'algorithme n'existe que pour des problèmes bien particuliers où on peut faire un choix localement optimal et où une solution optimale contient des solutions optimales pour des sous-problèmes.
- Pour prouver la correction de ces algorithmes, on pourra souvent se ramener à ces deux propriétés pour structurer la preuve.
- A la différence de la programmation dynamique, c'est une méthode top-down.

- **Localement optimale :** pas besoin de résoudre des sous-problèmes pour voir quel choix faire maintenant. Un petit regard suffit pour voir si c'est un bon choix.
    - **Par exemple :** Je sais tout de suite si une partie X fait partie du plus court chemin ou non (Dijkstra).
- **Top down** : par exemple, pour « rendre la monnaie » je regarde la somme la plus grande qui est inférieure ou égale à la somme et j'avance.
- **« Gloutons »**, car quand il mange, le glouton il ne rend pas ce qu'il a mangé.
    - **Donc,** quand on fait un choix : pas de retour en arrière.

Dijkstra
Prim, Kruskal
Codage de Huffman
...

} **Graphes**

Problème de l'allocation d'une ressource

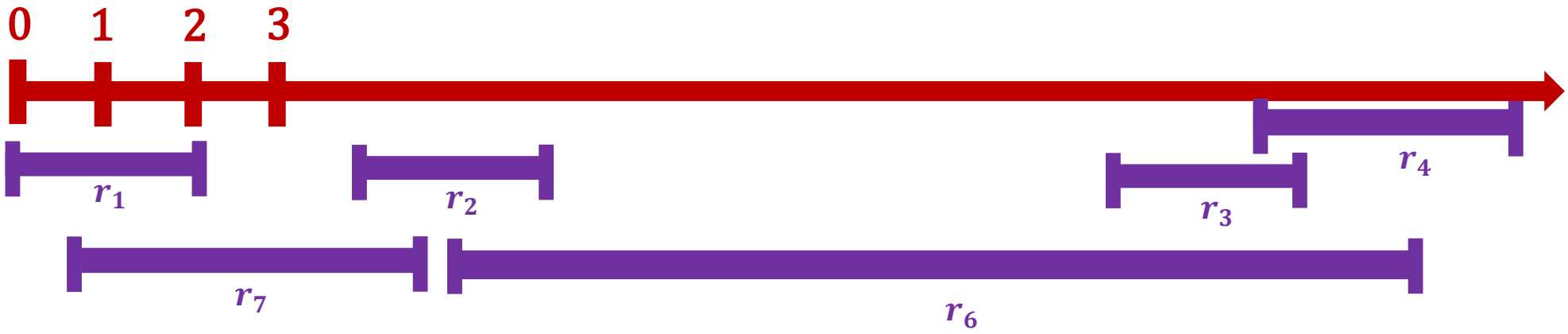
Une requête : (d, f) $\quad d < f \quad d, f \in \mathbb{N}$

Début — Fin

Input : un ensemble **E** de **n** requetes $\{(\mathbf{d_1}, \mathbf{f_1}), \dots, (\mathbf{d_n}, \mathbf{f_n})\}$

Output : un sous-ensemble de **E** compatible de taille max.

« Aucune requête n'en intersecte une autre … »

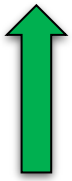

0
1
2
3
r_1
r_2
r_3
r_4
r_7
r_6

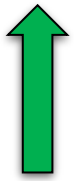
Définition :

Un ensemble de requêtes $\{(\mathbf{d_1}, \mathbf{f_1}), \ldots, (\mathbf{d_k}, \mathbf{f_k})\}$ est compatible

ssi $\forall \quad \mathbf{1} \leq \mathbf{i} \neq \mathbf{j} \leq \mathbf{k}$,

On a : $\mathbf{f_i} \leq \mathbf{d_j}$ ou $\mathbf{f_j} \leq \mathbf{d_i}$





« La requête *i* précède la requête *j* »

« La requête *j* précède la requête *i* »

Algo :

- Trier E par **date de fin croissante**.
- $\mathcal{F} = \emptyset$
- aux := 0

- Pour i = 1 à n :
  Si aux $\leq d_i$ Alors :
  $\mathcal{F} \mathrel{+}= \{(d_i, f_i)\}$
  aux := f_i

- Retourner $\mathcal{F}$

Preuve 1 : preuve *adhoc*

Supposons que l'algorithme ne marche pas, donc qu'il ne donne pas un résultat optimal.

- On va montrer que l'algorithme donne une solution optimale, c'est-à-dire un sous-ensemble $\mathcal{F}$ de taille maximale.
- Pour cela, on va supposer que ce n'est pas le cas et donc que la ou les solutions optimales contiennent plus de requêtes que le sous-ensemble renvoyé par l'algorithme.

Soit $\mathcal{F}$ le résultat envoyé par l'algorithme (le sous- ensemble compatible renvoyé par l'algorithme) :
$\mathcal{F} = \{(x_1, y_1) \dots (x_k, y_k)\}$ Dans l'ordre $y_1 < y_2 < .. < y_k$, c.-à-d. : trié par date de fin croissante.

Etant donnée la manière dont l'algorithme procède, on sait aussi que $(\mathbf{x_1}, \mathbf{y_1})$ est la première requête choisie par l'algorithme pour $\mathcal{F}$, $(\mathbf{x_2}, \mathbf{y_2})$ est la seconde, $(\mathbf{x_3}, \mathbf{y_3})$ la troisième,

Maintenant, je vais supposer que mon algorithme ce trompe.

Supposons que l'algo ne soit pas correct et qu'il existe une solution $\mathcal{G}$:

$\exists$ G un ensemble optimal = $(\mathcal{E}_1, t_1) \dots (\mathcal{E}_n, y_n)$

Preuve 1 : preuve *adhoc*

Dans cette supposition, $\boldsymbol{G}$ est une solution optimale avec $\boldsymbol{n}$ requêtes, $\boldsymbol{n > k}$

Parmi toutes les solutions optimales (Parmi les « $\boldsymbol{G}$ optimaux »), on ***choisit celle qui partage le plus des premiers choix fait par l'algorithme*** ($= \mathcal{F}$) , aussi triée par date de fin croissante.

Triée par date de fin croissante :

On a donc $\mathcal{E}_1 < t_1 \leq \mathcal{E}_2 \leq t_2 < \ldots$ ou $\mathcal{E}$ d'esigne la date de début de la requête et t sa date de fin.

$\boldsymbol{G}$ peut aussi s'écrire $\{(x_1, y_1) \ldots (x_i, y_i), (\mathcal{E}_{i+1}, t_{i+1}) \ldots (\mathcal{E}_n, t_n)\}$

et $(x_{i+1}, y_{i+1}) \neq (\mathcal{E}_{i+1}, t_{i+1})$: $\boldsymbol{i}$ est le nombre de premiers choix de l'algorithme partagés par $\boldsymbol{G}$. Et étant donnée hypothèse sur le choix de $\boldsymbol{\mathcal{F}}$, il n'existe pas de solutions optimales qui contiennent les requêtes $(\boldsymbol{x_1, y_1}) \ldots (\boldsymbol{x_{i+1}, y_{i+1}})$.

On sait aussi que $\boldsymbol{i < k}$ car si $\boldsymbol{i = k}$ alors l'algorithme aurait aussi essayer d'ajouter les requêtes $(\mathcal{E}_{i+2}, t_{i+2})$, $(\mathcal{E}_{i+2}, t_{i+2})$,. . . et que certaines d'entre elles auraient été ajoutées puisqu'elles sont compatibles avec les premières requêtes choisies. .

Preuve 1 : preuve *adhoc*

$$\begin{array}{c|c} Partie\ commune & la\ suite \\ \mathcal{F} = \{(x_1, y_1) \dots (x_i, y_i) & (x_{i+1}, y_{i+1}) \dots (x_k, y_k)\} \\ G = \{(x_1, y_1) \dots (x_i, y_i) & (\mathcal{E}_{i+1}, t_{i+1}) \dots (\mathcal{E}_n, t_n)\} \end{array}$$

Triée par date de fin croissante

Ordre des $\mathcal{F}$ ↗

$$0 \leq i < k$$

Ensuite je vais montrer qu'il y a une contradiction.

Si mon algorithme ne marche pas c'est que y'a des trucs optimaux plus grand de ceux trouvé par mon algorithme.

Je sais que mon algo a choisi (x_1, y_1) … puisque que mon algo n'est pas bon (« Supposons que l'algorithme ne marche pas, donc qu'il ne donne pas un résultat optimal »), je vais aller chercher un ensemble qui partage le plus avec l'ensemble optimal. Je vais montrer que je peux toujours ajouter dans l'ensemble optimal l'ensemble pris par mon algorithme.

Preuve 1 : preuve *adhoc*

Maintenant on définit $\boldsymbol{G'}$ par $G \setminus \{(\mathcal{E}_{i+1}, t_{i+1})\} \cup \{(x_{i+1}, y_{i+1})\}$.

On voit alors que $\boldsymbol{G'}$ est un sous-ensemble compatible puisque si l'algorithme choisit la requête (x_{i+1}, y_{i+1}), c'est qu'elle est la requête ayant une date de fin minimale parmi celles compatibles avec $(x_1, y_1) \dots (x_i, y_i)$,

et donc la requête $(\mathcal{E}_{i+1}, t_{i+1})$ a une date de fin supérieure ou égale à celle de (x_{i+1}, y_{i+1}), et donc remplacer $(\mathcal{E}_{i+1}, t_{i+1})$ par (x_{i+1}, y_{i+1}) ne pose pas de problème pour les autres requêtes de $\boldsymbol{G}$: on a bien $y_{i+1} \leq \mathcal{E}_{i+2}$ (car $y_{i+1} \leq t_{i+1}$, car Tri des requêtes !).

$\boldsymbol{G'}$ est donc un sous-ensemble compatible et de même taille que G, c'est donc une solution optimale et il partage un premier choix de l'algorithme de plus que $\boldsymbol{G}$, on a donc une contradiction avec les hypothèses de départ.

Preuve 2 : en repartant des deux propriétés des algorithmes gloutons

Pour cette seconde preuve, on procède en montrant d'abord que l'algorithme fait un « bon » premier choix, c'est à dire un choix qui permettra *in fine* d'obtenir un sous-ensemble optimal : autrement dit, un « bon choix » est un choix inclus dans une solution optimale.

Proposition 1 : « bon premier choix »

Soit $\mathbf{E}$ un ensemble de requêtes,

soit $\mathbf{e} = (\mathbf{d}, \mathbf{f})$ une requête de $\mathbf{E}$ ayant une date de fin minimale,

Alors : il existe une solution* contenant $\mathbf{e}$.

Un ensemble compatible de taille max...

Proposition 1 dit que mon algo ne se trompe pas

Comment prouver ça ?

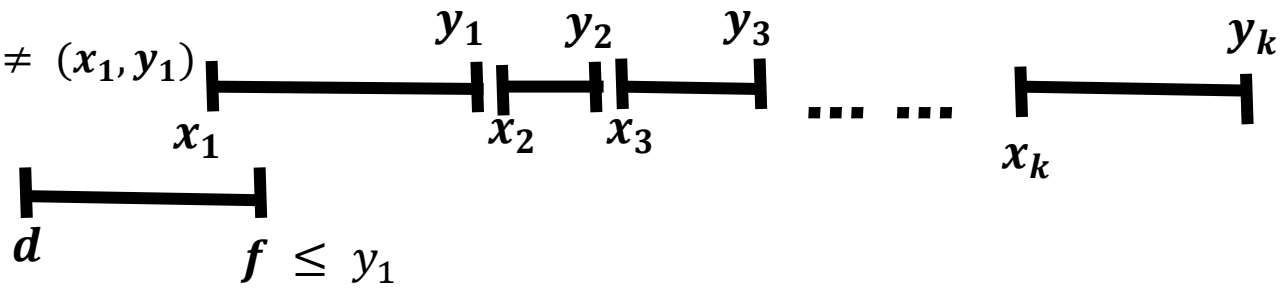
Soit G une solution = $\{(x_1, y_1) \dots (x_k, y_k)\}$ triée par date de fin croissante, donc telle que $y_1 \leq y_2 \leq .. \leq y_k$ ($x_i < y_1 \leq y_{i+1}$..).

Preuve 2 : en repartant des deux propriétés des algorithmes gloutons

Maintenant, il y a 2 cas :

- Soit $\boldsymbol{e = (d, f) = (x_1, y_1)}$
  c'est bon, on a fini ; on a une solution optimale qui contient e

- Soit $\boldsymbol{e = (d, f) \neq (x_1, y_1)}$



Soit $G' = G \setminus \{(x1, x2\} \cup \{(d, f)\}$
→ G' est compatible & optimale.

Etant donne l'algorithme, on sait que f ≤ y1, d'où f ≤ x2 et donc le sous-ensemble $\{e, (x_2, y_2) \ldots (x_k, y_k)\}$ est compatible et constitue aussi une solution optimale qui contient le premier choix de l'algorithme.

Preuve 2 : en repartant des deux propriétés des algorithmes gloutons

Sous-ensemble optimale

Proposition 2 :

- Soit $\mathbf{E}$ un ensemble de requêtes,
- Soit $\mathbf{G}$ une **solution optimale*** pour $\mathbf{E}$,
- Soit $\mathbf{e} \in \mathbf{G}$,

→ « Sous – ensemble compatible de taille max »

Alors :

$\mathbf{G'} = \mathbf{G} \backslash \{\mathbf{e}\}$ est une **solution optimale** pour l'ensemble des requêtes de $\mathbf{E}$ compatibles avec $\mathbf{e}$. → Les requêtes qui n'intersectés pas *e*

Alors : il existe une solution* contenant $\mathbf{e}$.

C'est-à-dire pour l'ensemble de requêtes :

$$E' = \{e' \in E \mid d' \geq f \vee d \geq f'$$

Preuve 2 : en repartant des deux propriétés des algorithmes gloutons

Pourquoi proposition 2 est vrai ?

C'est vrai car imaginer que ça soit faux...

Supposons le contraire : supposons que la proposition est fausse.

Donc, il existe G'' ($\exists\, G''$) Une solution optimale pour

« E restreintes (limités) aux requêtes compatibles avec e », avec $|G''| > |G'|$ →

il suffit de prendre $G'' \cup \{e\}$ pour obtenir un sous-ensemble compatible

(car par hypothèse G'' est compatible avec e).

L'ensemble $G'' \cup \{e\}$ est compatible et $|G'' \cup \{e\}| > |G|$!!

$|G'' \cup \{e\}| > |G|$ ⇒ G est pas optimal

Et la y'a une contradiction car G était soi-disant optimal.

Preuve 2 : en repartant des deux propriétés des algorithmes gloutons

« L'algorithme glouton est correct »

Il reste à en tirer le théorème de correction…

Pour cela il faut noter qu'une itération de l'algorithme consiste à choisir une requête ayant une date de fin minimale puis à passer à l'étape suivante où il va chercher une solution pour l'ensemble des requêtes compatibles avec son choix précédent, etc.

La preuve de correction se fait alors par induction sur la taille n de E :

Induction sur $|\mathbf{E}|$:

- **h.i. (hypothèse d'Induction)** L'algo est correct pour les ensembles de requêtes de taille < n

- $\underline{\boldsymbol{n} = |\mathbf{E}| = \mathbf{1}}$ **:** l'algorithme renvoie l'unique requête et c'est bien évidemment la solution optimale !

Preuve 2 : en repartant des deux propriétés des algorithmes gloutons

- $\underline{n = |E| > 1 \quad (n+1)}$:

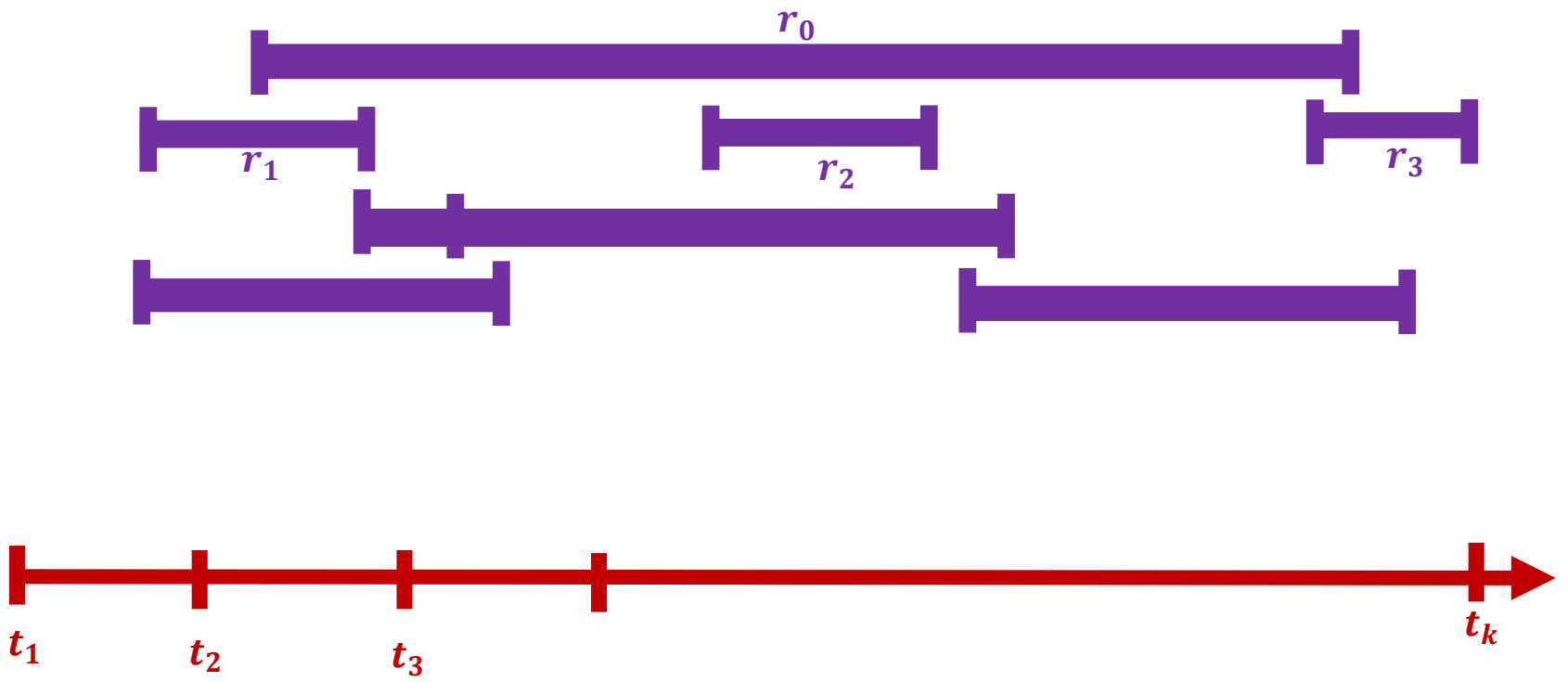
Soit $\mathcal{F}$ la solution renvoyée par l'algorithme et soit e la première requête choisie par l'algorithme : $e = (d, f)$ la requête de date f_{min}

  - **Proposition 1 :** On sait qu'il existe une solution optimale G qui contient (d, f).
  - **Proposition 2 :** On sait que $G' = G \setminus \{d, f\}$ est une solution optimale pour les requêtes de E compatible avec (d, f),

➔ Cet ensemble est plus petit que E !

➔ Et l'algo (par h.i.) donne une solution de la bonne taille.

Et si on veut maximiser la durée totale de location ?



.