Algorithmes d'Optimisation

Stéphane Perennes

DUT INFO - IUT Nice Côte d'Azur

15 mars 2022

Problèmes d'optimisation combinatoire : définition

Problème défini par :

- un ensemble discret de **solutions** (souvent $2^{\Theta(n)}$ solutions.)
- une **fonction objectif** (à maximiser ou minimiser)
- certaines solutions sont dites **réalisables** (i.e. valides).

Trouver une solution réalisable qui minimise (ou maximise) la fonction objectif ?

Problèmes d'optimisation combinatoire : condition supplémentaire

Plus formellement

- On peut vérifier une solution *facilement*.
- **en d'autres termes** : Il existe un algorithme Polynomial (temps $\Theta(n^k)$) qui vérifie la validité d'une solution et qui calcule son coût.

Formellement le problème appartient à la classe famille *NP*. [1]

1. voir cours de complexité

Problème d'optimisation combinatoire : Exemples

- Plus *court* chemin entre deux sommets a, b d'un graphe. (Routage, GPS)
  - Solutions : ensemble d'arêtes, pour n arêtes 2^n solutions.
  - Validité : les arêtes relient a et b
  - Objectif, coût : somme des poids des arêtes.
- Arbre Couvrant de Poids minimum (Minimum Spanning Tree).
  - Solutions : ensemble d'arêtes S, pour n arêtes 2^n solutions.
  - Toute pair de sommet est reliée via S
  - Objectif, coût : somme des poids des arêtes.

Problèmes d'opti. comb. : Exemple du Sac à Dos

Remplissage de sac à dos (Knapsack)

- Ensemble d'objets $o \in O$ dotés d'un volume v_o et d'un bénéfice b_o.
- Sac de volume maximum V.
- Déterminer le meilleur sac.

Dans notre cadre :

- Solutions : S sous ensemble d'objets
- n objets 2^n solutions.
- Validité : $\sum_{s \in S} v_s \leq V$.
- Objectif : ici le Bénéfice : $\sum_{s \in S} b_s$.

Problème d'optimisation combinatoire : exemple du TSP

Traveling Salesman Problem TSP

- ensemble de villes à parcourir
- matrice des distances entre les villes

Quel est le circuit[a] le plus court qui les parcourt ?

---

a. point de départ identique au point d'arrivée

Problèmes d'opti. comb. : Taxonomie/Dichotomie

Problèmes Faciles/Polynomiaux

- Problèmes pour lesquels on connait un algorithme rapide (en temps $\Theta(n^k)$ avec k fini et fixé)
- En général on cherche alors à minimiser k et à déterminer la complexité exacte.
- Problèmes considérés comme classiques $\rightarrow$ bibliotèques optimisées vis à vis des constantes cachées.

Problèmes dits NP-Complets

- On ne connait pas de technique efficace.
- On a montré que si une telle technique existait on saurait résoudre "tous" les problemes rapidement (pex. TSP).
- On cherche alors de méthodes approchées, des heuristiques.

Force brute

Solution évidente : la force brute

En algorithmique, la force brute (recherche exhaustive) consiste à parcourir toutes les solutions afin de trouver la meilleure.

Complexité

Dans le cas du TSP, il y a $O((n-1)!)$ solutions possibles.
$17! \sim 355000$ Milliards. Au dela de 17 cette approche est pratiquement impossible même en traitant 10^9

Quelques techniques de résolution

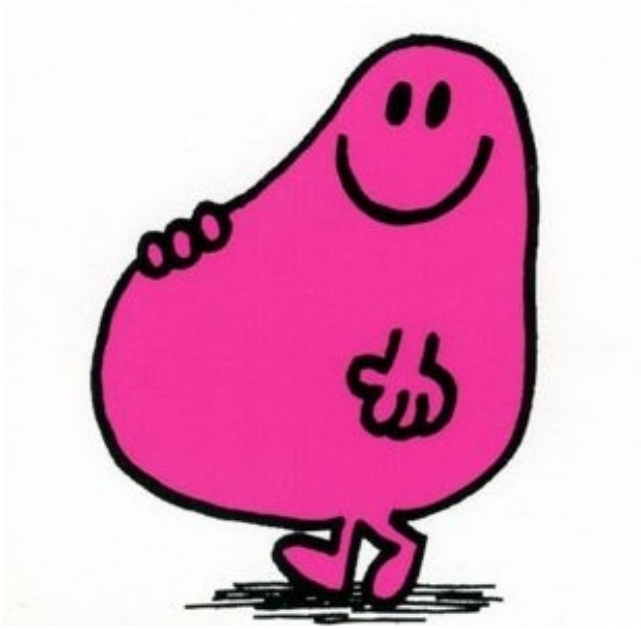
De nombreuses techniques de résolution.

- Méthode Gloutonne.
- Programmation dynamique.
- Programmation Linéaire, arrondis relaxation.
- Heuristique avec des garanties prouvées

Alternative gloutonne

Un algorithme est dit glouton [2] si :

- il construit une solution étape par étape (avec des étapes courtes)
- sans retour en arrière
- On augmente la solution courante au "moindre coût".
- en optimisant un des critères du problème seulement



Il n'y a pas de définition formelle mais on peut dire que c'est un algorithme à décision locale

1. on parle aussi d'heuristique gloutonne

Utilisation

En général ce type d'algorithme ne permet pas d'obtenir la solution optimale, mais :

- on trouve une solution
- rapidement (temps non exponentiel)
- on peut obtenir parfois une approximation bornée (on sait mesurer l'écart avec la solution optimale)
- la solution peut être améliorée ensuite dans un algorithme plus complexe

Et parfois on obtient même la solution optimale !

Exemple 1 : L'arbre couvrant de coût Minimum (MST [a])

a. Minimum Weight Spanning Tree.

Entrée :

- Un Graphe connexe avec des sommets V et des arêtes E.
- Une fonction de poids w sur les arêtes, $w(e)$.

Sortie :

- Un arbre couvrant de poind minimum

Algorithme de Prim (Jarnik 1930, Prim 1957)

W ensemble des sommets couverts, T arbre courant couvrant W

- $W = \{v_0\}$, $T = \emptyset$.
- Tant que $W \neq V$
  - Trouver une arête e de poids minimum sortant de W.
  - Ajouter e à T : $T = T \cup \{e\}$.
  - si $e = (w_0, a), w_0 \in W$ faire $W = W \cup \{a\}$.

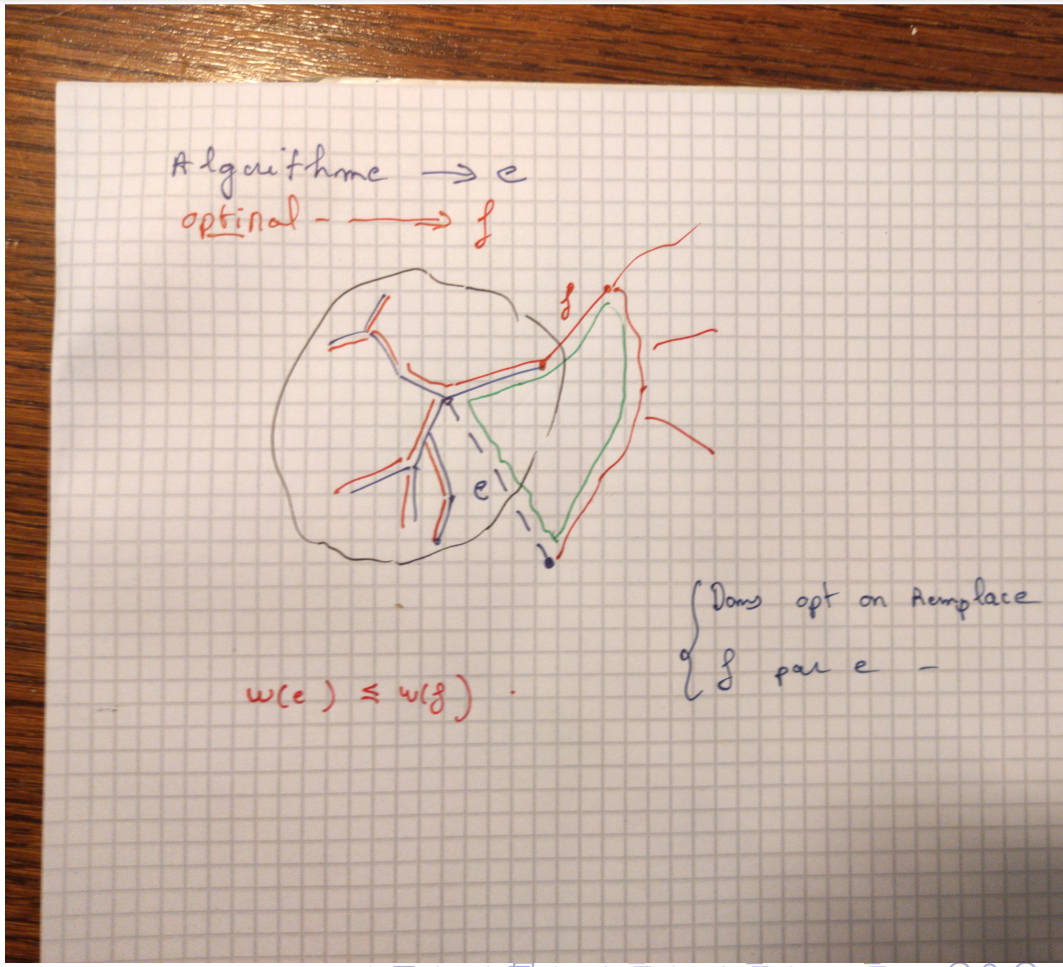
Aspect Glouton ?

- On augmente pas à pas la taille de l'arbre pour un coût minimum.
- À chaque étape on ajoute 1 sommet et 1 arête.
- Pas de remise en cause des choix.

Prim est optimal

Preuve

- On regarde au moment ou l'algorithme devie de l'optimal
- Algo choisi e, Opt choisi f



Implémentation ?

Choisir la bonne structure de donnée

À chaque sommet $u \notin W$ on associe un index $i(v) = \{$le poids de l arête la plus légère le connectant à $V\}$. On utilise une structure d'index Q.

- (1) Extraire l'index minimum et supprimer l'élément associé v_0
- (2) Pour chaque voisin u de v_0 qui est dans Q faire $i(v_0) = min(i(v_0), w(v_0, u))$

On va avoir $|V|$ extractions et $|E|$ mises à jour de clef.

Fibonacci Heap

FindMin	$\Theta(1)$
DeleteMin	$\Theta(\log n)$
DecreaseKey	$\Theta(1)$

Donc $\Theta(|V| \log |V|) + \Theta(|E|)$.

Couverture Gloutonne

Problème de couverture

Données :

- Ensemble de **base** S (ground set).
- Familles de sous ensembles $S_0, S_1, S_i, \ldots S_m$.

Objectif : Trouver la plus petit sous famille dont l'union est S.

$$Min|I|, \cup_{i \in I} S_i = S$$

Application : Couverture réseau sans fils, desserte (placement de centre de service, ...).

Couverture Gloutonne

Methode Gloutonne : Itérativement on choisit l'ensemble qui couvre le plus de sommets.

Algorithme Glouton

$I = \emptyset$ (solution) , $W = V$ (sommets à couvrir)

- tant que $W \neq \emptyset$
  - Trouver i_0 tel que $|S_{i_0} \cap W|$ soit maximal : l'ensemble qui couvre le plus de sommets.
  - $I = I \cup \{i_0\}, W = W \setminus S_{i_0}$.

Efficacité de la Couverture Gloutonne

Si $|S| = n$

$$cout(Algo) \leq \log 2(n) \times cout(Opt)$$

Argument

Quand il reste un ensemble W d'éléments à couvrir alors $max_{i \in I} |S_i \cap W| \geq |W|/Opt$

Conséquence

Tant que $W \geq \frac{n}{2}$ $max_{i \in I} |S_i \cap W| \geq \frac{n}{2Opt}$

Efficacité de la Couverture Gloutonne bis

Si $|S| = n$

$$cout(Algo) \leq \log 2(n) \times cout(Opt)$$

Consequence

- Tant que $W \geq \frac{n}{2}$ on couvre au moins $\frac{n}{2Opt}$ nouveaux éléments à chaque étapes.
- Pour passer de $|W| = n$ à $|W| \leq n/2$, au plus Opt étapes,

On a donc :

$$Algo(n) \leq Opt + Algo(n/2)$$
$$Algo(n) \leq Opt + Opt + Algo(n/4)$$
$$Algo(n) \leq Opt + Opt + Opt + Algo(n/8)$$
$$Algo(n) \leq \log(n)Opt + Algo(n/2^{log2(n)}) = \log(n)Opt + \Theta(1)$$

Couverture Pondérée

Les ensembles $S_i, i \in I$ on maitenant un coût non uniforme c_i(et non pas 1.

Efficacité d'un ensemble :

$$Eff(S_i) = \frac{Gain}{Cout} = \frac{|S_i \cap W|}{c_i}$$

Algo Glouton : Choisir l'ensemble le plus efficace.

Qualité de la solution

On peut montrer que

$$cout(Algo) \leq (1 + \frac{1}{2} + \cdots + \frac{1}{n})Opt \sim \ln(n)Opt$$

Exemple 1 : ordonnancement d'intervalles

On considère :

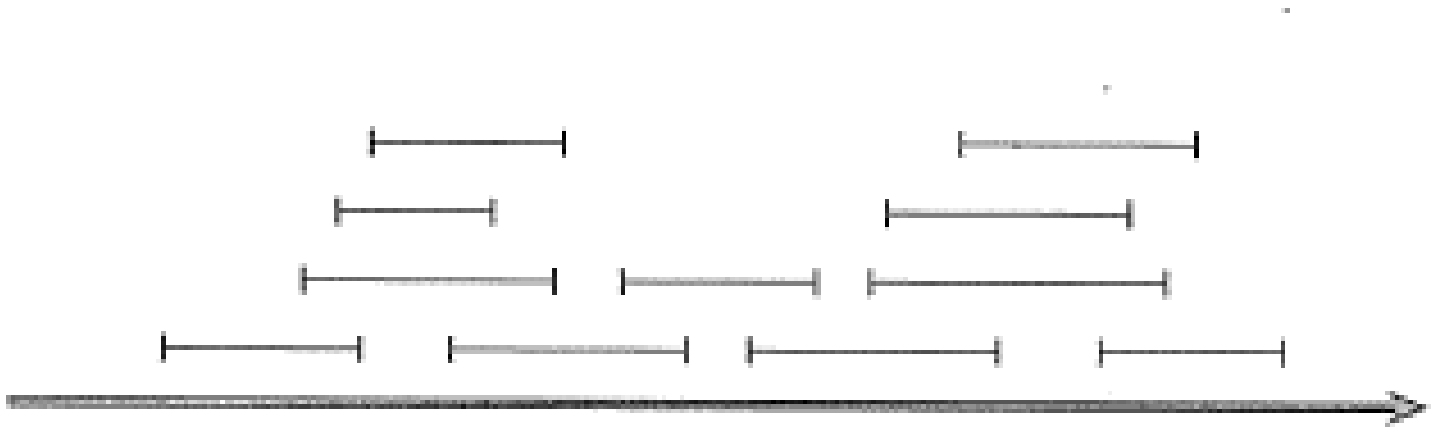
- un ensemble de requêtes $R = \{1, 2, \cdots, n\}$
- la requête i correspond à un intervalle de temps qui débute à la date $s(i)$ et finit à la date $f(i)$
- un sous ensemble de requêtes est valide si aucune paire de requêtes a une intersection non vide
- le problème est de trouver un sous ensemble de requêtes optimal (dont la cardinalité est maximale)

On pourrait chercher toutes les solutions mais dès que le nombre de requêtes n est grand, aucun ordinateur ne peut parcourir toutes les solutions dans un temps à l'échelle humaine : $O(2^n)$.

Ce problème est repris de J. Kleinberg and E. Tardos. *Algorithm Design.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

Exemple 1 : ordonnancement d'intervalles

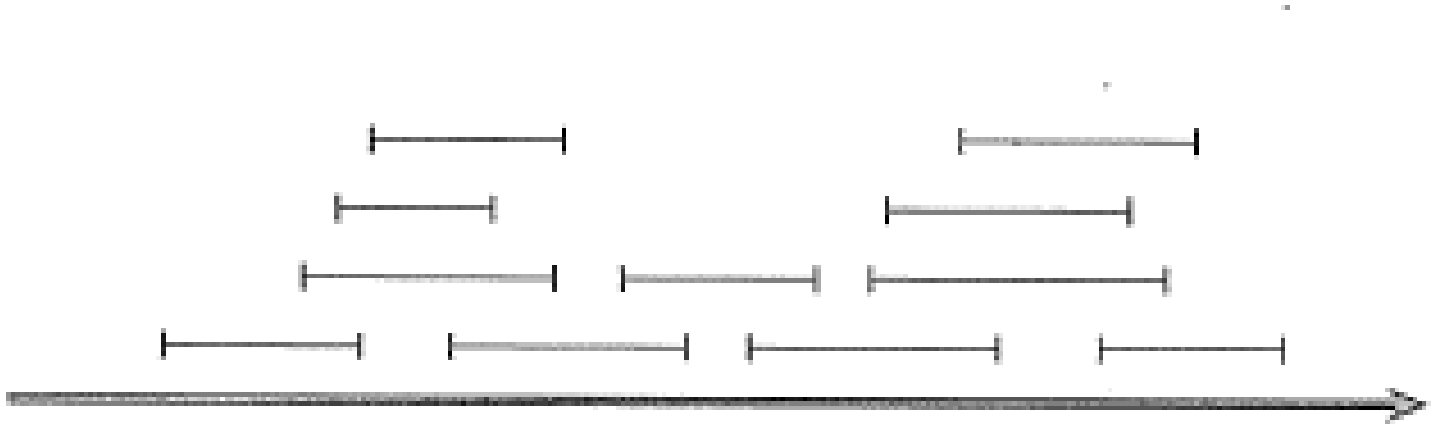
Exemple à 11 requêtes et une seule solution de taille 4 (les 4 sur la ligne du bas) :



Algorithme glouton pour l'ordonnancement d'intervalles

Algorithme / heuristique 1

sélectionner les requêtes qui débutent le plus tôt d'abord



Algorithme glouton pour l'ordonnancement d'intervalles

Algorithme / heuristique 1

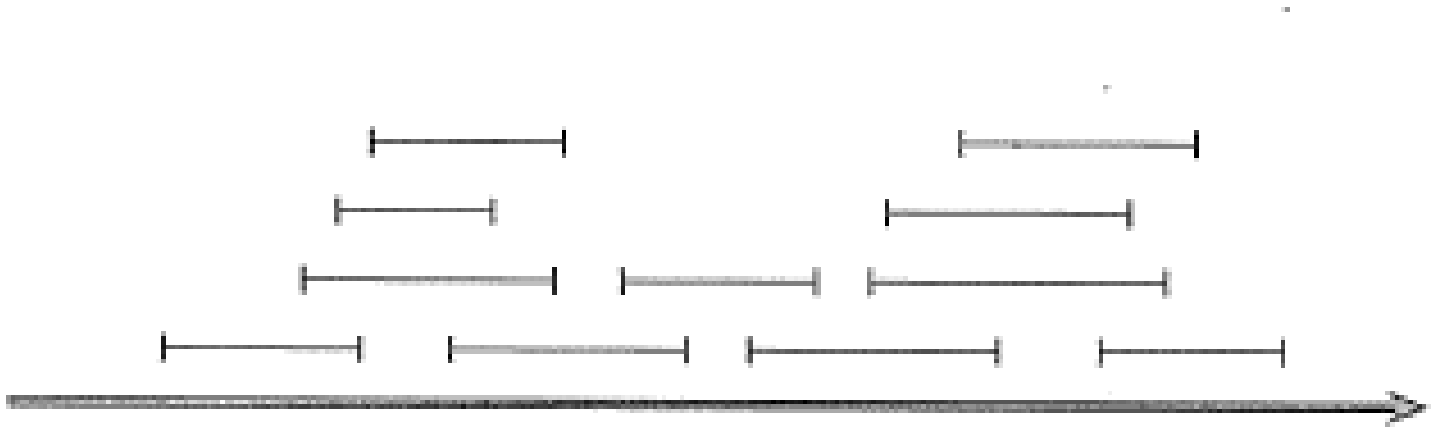
sélectionner les requêtes qui débutent le plus tôt d'abord

Pas optimal, la preuve :

Algorithme glouton pour l'ordonnancement d'intervalles

Algorithme / heuristique 2

sélectionner les plus petites requêtes $(f(i) - s(i))$ d'abord

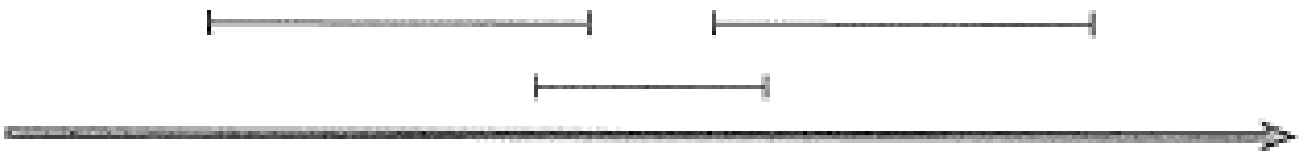


Algorithme glouton pour l'ordonnancement d'intervalles

Algorithme / heuristique 2

sélectionner les plus petites requêtes $(f(i) - s(i))$ d'abord

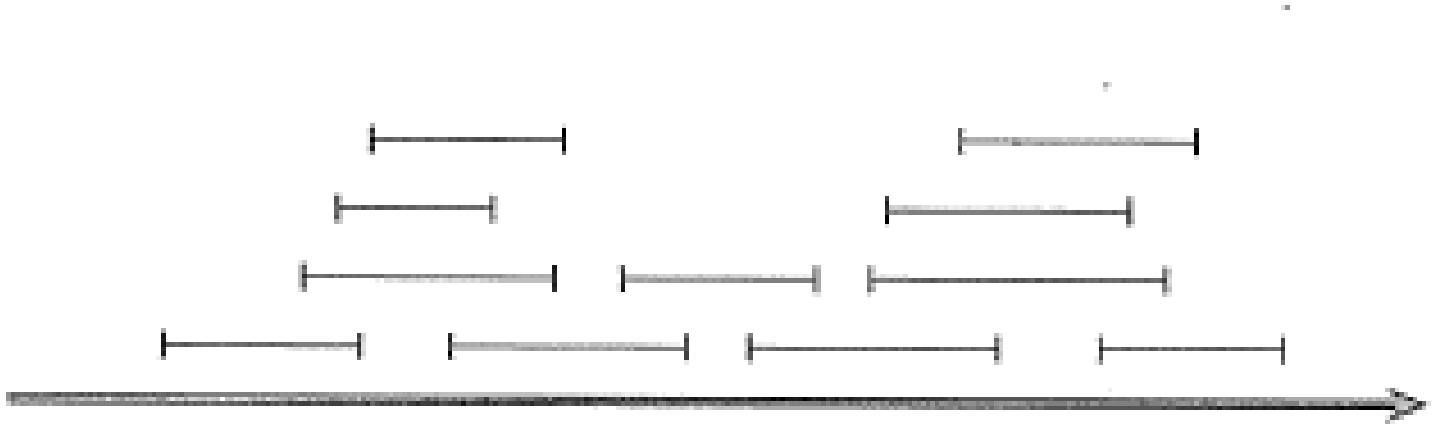
Pas optimal, la preuve :



Algorithme glouton pour l'ordonnancement d'intervalles

Algorithme / heuristique 3

sélectionner les requêtes qui ont le moins de conflits d'abord

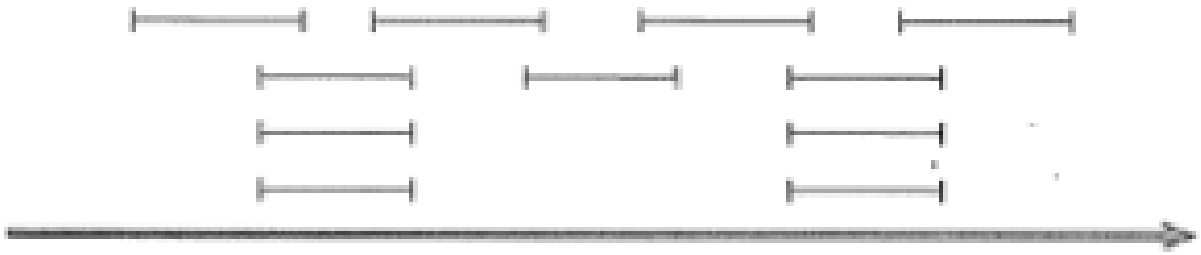


Algorithme glouton pour l'ordonnancement d'intervalles

Algorithme / heuristique 3

sélectionner les requêtes qui ont le moins de conflits d'abord

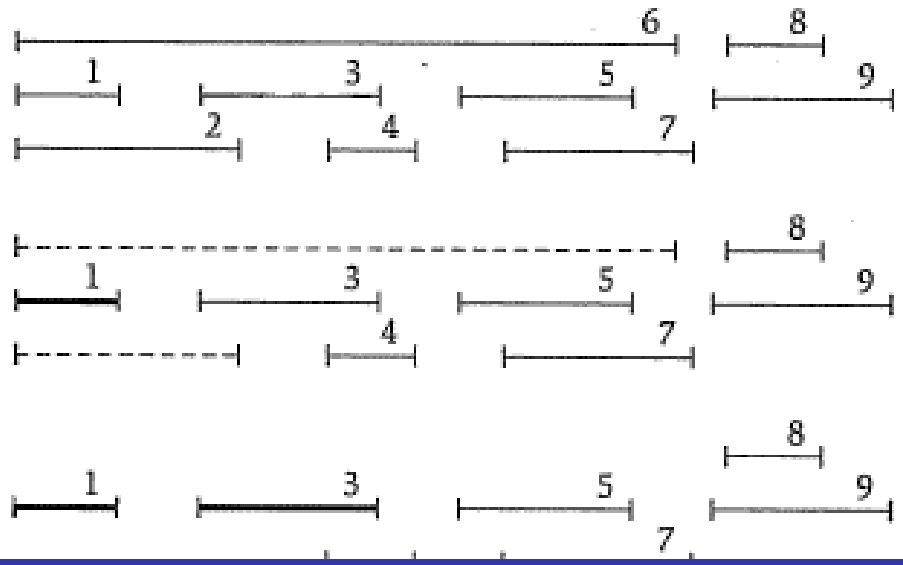
Pas optimal, la preuve :



Algorithme glouton pour l'ordonnancement d'intervalles

Algorithme / heuristique 4

sélectionner les requêtes qui finissent en premier d'abord



Optimalité

L'algorithme 4 est optimal

- soit $A = \{i_1, i_2, \cdots, i_k\}$ avec $|A| = k$ l'ensemble résultat de l'algorithme 4
- L'ensemble A construit étape par étape est valide (pas d'intersection entre requêtes par construction)
- Soit $\mathcal{O}$ un ensemble d'intervalles optimal, il suffit de prouver que $|A| = |\mathcal{O}|$ (il n'y a pas forcément unicité)
- Si $\mathcal{O} = \{j_1, j_2, \cdots, j_m\}$, alors on veut prouver que $k = m$
- A et $\mathcal{O}$ sont ordonnés (tant pour $s(i)$ que $f(i)$ puisque les requêtes sont compatibles)

Principe de la preuve :

à chaque étape, l'algorithme 4 $\ll$ reste devant $\gg$!

Optimalité

Les intervalles de A finissent au moins aussi tôt que ceux de $\mathcal{O}$:

$$\forall r < k, f(i_r) \leq f(j_r) \tag{1}$$

Preuve par induction :

- Pour $r = 1$ (1) est vraie : l'algorithme 4 choisit la requête qui finit le plus tôt en premier
- Pour $r > 1$: on suppose que (1) est vraie pour $r - 1$ et on va la prouver pour r.
  Si i_r termine après j_r alors que i_{r-1} avait terminé avant j_{r-1}, on aurait du choisir j_r à la place de i_r qui était disponible et compatible.

Optimalité

L'ensemble A est optimal.

Preuve par contradiction :

- Si A n'est pas optimal, alors pour tout $\mathcal{O}$ on a $m > k$
- Si on applique (1) avec $r = k$ on obtient $f(i_k) \leq f(j_k)$
- Si $m > k$ il existe une requête j_{k+1}
- La requête j_{k+1} débute après j_k et débute aussi après la fin de i_k
- Après avoir supprimé toutes les requêtes non compatibles avec $i_1, \cdots, i_k$ il reste j_{k+1} dans R, ce qui est en contradiction avec l'algorithme 4 qui termine quand R est vide.

Conclusion :

L'algorithme 4 est optimal !

Complexité

Implémentation de l'algorithme 4 :

- trier R par date de fin : $O(n \log n)$
- appliquer l'algorithme glouton : $O(n)$

Conclusion : l'algorithme 4 est en $O(n \log n)$

Autres exemples où le glouton est optimal

- Plus courts chemins (Dijkstra)
- Arbres couvrants de poids minimum (Kruskal ou Prim)
  - Kruskal : on ajoute les arêtes une à une sauf si cela crée un cycle
  - Prim : on traite les nœuds un à un et on met à jour les voisins du nœud choisi

Problème du rendu de monnaie (coin changing)

Un caissier doit rendre la monnaie au client en utilisant le moins de pièces/billets possible.

- On dispose des valeurs 10, 5, 2 et 1 en nombre illimité
- On ajoute la plus grosse valeur possible à chaque étape (sans dépasser le total)

Est-ce que cet algorithme glouton est optimal ?

Problème du sac à dos (knapsack)

Algorithmes gloutons possibles :

- article de plus grande valeur d'abord
- article de plus grand rapport valeur/poids d'abord

Présentation détaillée du problème

Autres problèmes sans glouton optimal

- Voyageur de commerce (TSP)
- Satisfaisabilité (rendre vraie) d'une formule de la logique propositionnelle (SAT, 3-SAT). Exemple de formule donnée en forme normale conjonctive avec trois littéraux par clause : $(x \vee y \vee \neg z) \wedge (x \vee \neg w \vee z) \wedge \cdots$
- Bin packing
- Partitionnement
- ...