

DIU Enseignement de l'Info au Lycée

Cours 2 : Algorithmes gloutons

Université de Montpellier
2018 – 2019

Les méthodes gloutonnes

- ▶ En un mot, **faire à chaque étape le choix possible qui semble le meilleur.**

Parfois, ça marche et on obtient des bons algos...

- ▶ On voit dans ce Cours 2 des algorithmes gloutons simples et dont on peut prouver l'optimalité (d'autres seront présentés dans le Cours 4).
- ▶ Algos gloutons au programme de l'option NSI en 1ère !

1. Exemple 1 : choix de cours
2. Qu'est qu'un algorithme glouton ?
3. Exemple 2 : le sac-à-dos (fractionnaire)
4. Exemple spécial : approximation pour SETCOVER dans le plan

1. Exemple 1 : choix de cours

2. Qu'est qu'un algorithme glouton ?

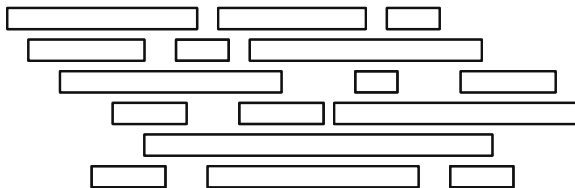
3. Exemple 2 : le sac-à-dos (fractionnaire)

4. Exemple spécial : approximation pour SETCOVER dans le plan

Définition du problème

Entrée un ensemble \mathcal{C} de cours $C_i = (d_i, f_i)$ [début, fin], $i = 1, \dots, n$

Sortie un ensemble ordonné maximal de cours $(C_{i_1}, \dots, C_{i_k})$ tels que pour tout $j < k$, $f_{i_j} \leq d_{i_{j+1}} \rightsquigarrow$ cours **compatibles**

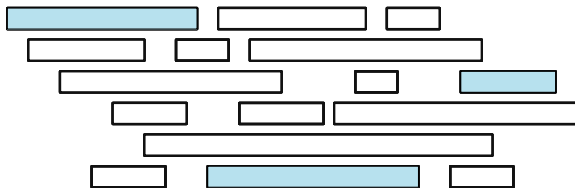


modifié d'après *Algorithms* de J. Erickson

Définition du problème

Entrée un ensemble \mathcal{C} de cours $C_i = (d_i, f_i)$ [début, fin], $i = 1, \dots, n$

Sortie un ensemble ordonné maximal de cours $(C_{i_1}, \dots, C_{i_k})$ tels que pour tout $j < k$, $f_{i_j} \leq d_{i_{j+1}} \rightsquigarrow$ cours **compatibles**

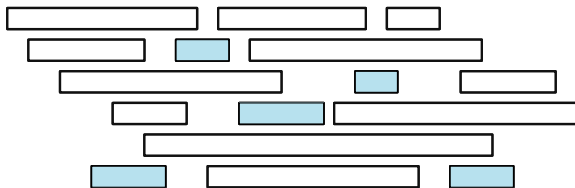


modifié d'après *Algorithms* de J. Erickson

Définition du problème

Entrée un ensemble \mathcal{C} de cours $C_i = (d_i, f_i)$ [début, fin], $i = 1, \dots, n$

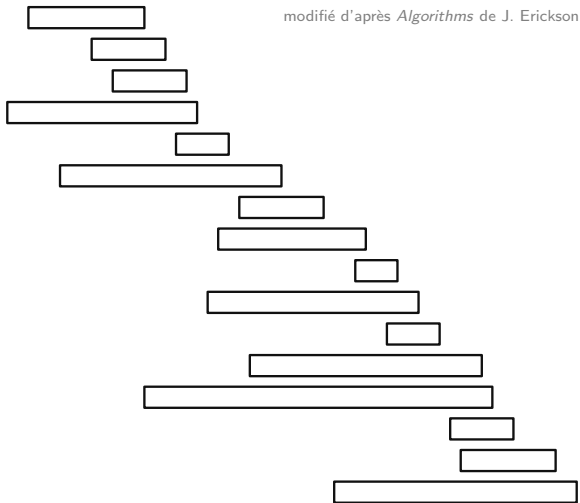
Sortie un ensemble ordonné maximal de cours $(C_{i_1}, \dots, C_{i_k})$ tels que pour tout $j < k$, $f_{i_j} \leq d_{i_{j+1}} \rightsquigarrow$ cours **compatibles**



modifié d'après *Algorithms* de J. Erickson

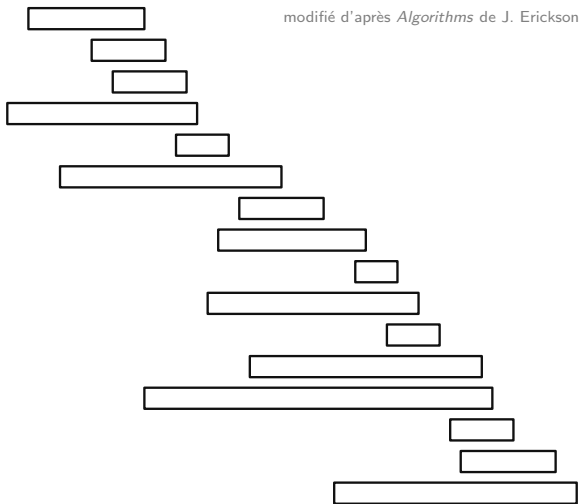
Idée gloutonne

- Tri des cours par **dates de fin croissantes**



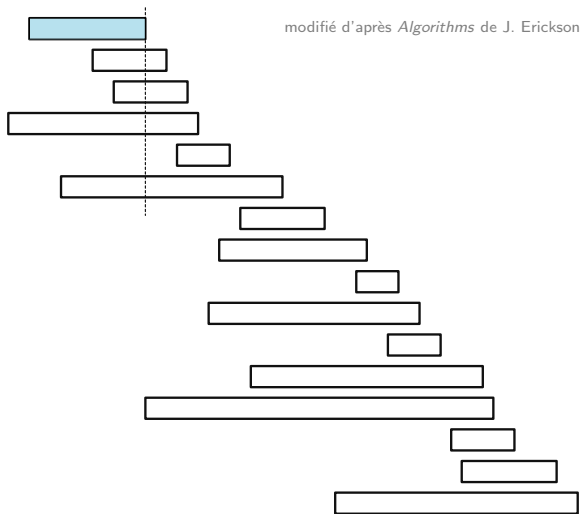
Idée gloutonne

- ▶ Tri des cours par **dates de fin croissantes**
- ▶ Choix *glouton* : sélectionner le cours qui finit **le plus tôt**



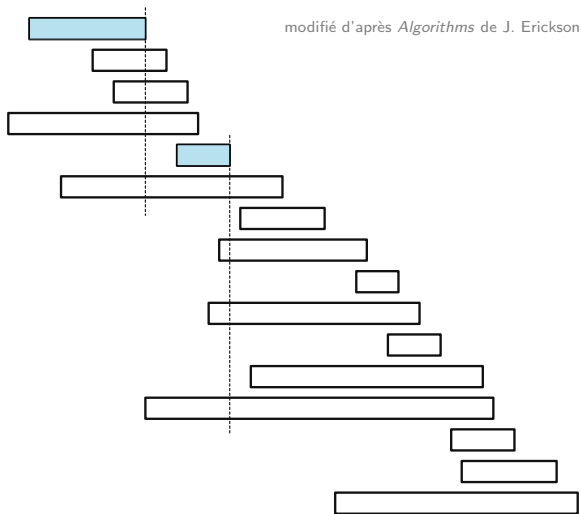
Idée gloutonne

- ▶ Tri des cours par **dates de fin croissantes**
- ▶ Choix *glouton* : sélectionner le cours qui finit **le plus tôt**



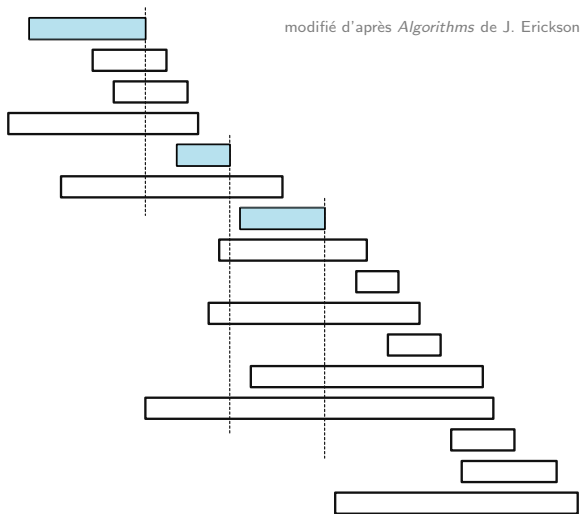
Idée gloutonne

- ▶ Tri des cours par **dates de fin croissantes**
- ▶ Choix *glouton* : sélectionner le cours qui finit **le plus tôt**



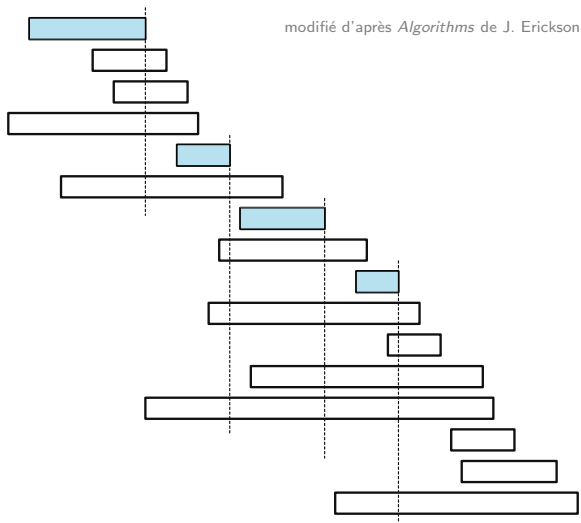
Idée gloutonne

- ▶ Tri des cours par **dates de fin croissantes**
- ▶ Choix *glouton* : sélectionner le cours qui finit **le plus tôt**



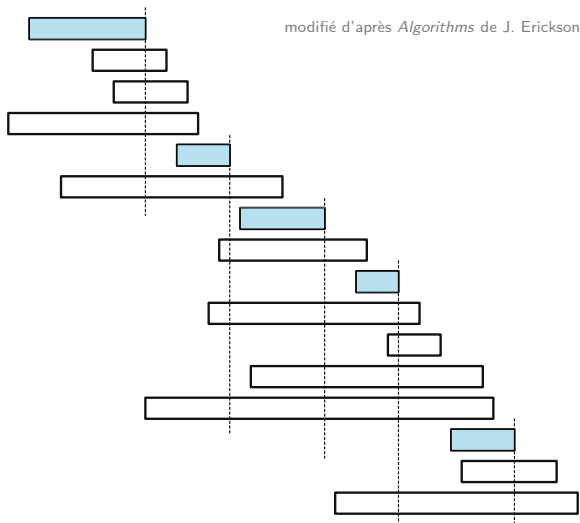
Idée gloutonne

- ▶ Tri des cours par **dates de fin croissantes**
- ▶ Choix *glouton* : sélectionner le cours qui finit **le plus tôt**



Idée gloutonne

- ▶ Tri des cours par **dates de fin croissantes**
- ▶ Choix *glouton* : sélectionner le cours qui finit **le plus tôt**



Algorithme glouton

Algorithme : CHOIXCOURSGLOUTON(C)

Trier C en fonction des fins

$I \leftarrow \{C[1]\}$ // Cours choisis

$f \leftarrow \text{FIN}(C[1])$ // Fin du dernier cours choisi

pour $i = 2$ à n **faire**

si $\text{DÉBUT}(C[i]) \geq f$ **alors**

$I \leftarrow I \cup \{C[i]\}$

$f \leftarrow \text{FIN}(C[i])$

retourner I

Algorithme glouton

Algorithme : CHOIXCOURSGLOUTON(C)

Trier C en fonction des fins

$I \leftarrow \{C[1]\}$ // Cours choisis

$f \leftarrow \text{FIN}(C[1])$ // Fin du dernier cours choisi

pour $i = 2$ à n **faire**

si $\text{DÉBUT}(C[i]) \geq f$ **alors**

$I \leftarrow I \cup \{C[i]\}$

$f \leftarrow \text{FIN}(C[i])$

retourner I

Question

Quelle est la complexité de CHOIXCOURSGLOUTON ?

Algorithme glouton

Algorithme : CHOIXCOURSGLOUTON(C)

Trier C en fonction des fins

$I \leftarrow \{C[1]\}$ // Cours choisis

$f \leftarrow \text{FIN}(C[1])$ // Fin du dernier cours choisi

pour $i = 2$ à n **faire**

si $\text{DÉBUT}(C[i]) \geq f$ **alors**

$I \leftarrow I \cup \{C[i]\}$

$f \leftarrow \text{FIN}(C[i])$

retourner I

Question

Quelle est la complexité de CHOIXCOURSGLOUTON ?

C'est le tri le plus coûteux (voir Chap. 3) $\rightsquigarrow O(n \log n)$

Validité de l'algorithme

Théorème

CHOIXCOURSGLOUTON est *optimal* : il renvoie un ensemble maximal (d'indices) de cours compatibles, c-à-d qu'il n'existe pas d'ensemble strictement plus grand de cours compatibles.

Preuve

Validité de l'algorithme

Théorème

CHOIXCOURSGLOUTON est *optimal* : il renvoie un ensemble maximal (d'indices) de cours compatibles, c-à-d qu'il n'existe pas d'ensemble strictement plus grand de cours compatibles.

Preuve

- Notons $\mathcal{C} = (C_1, \dots, C_n)$ les cours triés par dates de fin croissantes.

Validité de l'algorithme

Théorème

CHOIXCOURSGLOUTON est *optimal* : il renvoie un ensemble maximal (d'indices) de cours compatibles, c-à-d qu'il n'existe pas d'ensemble strictement plus grand de cours compatibles.

Preuve

- ▶ Notons $\mathcal{C} = (C_1, \dots, C_n)$ les cours triés par dates de fin croissantes.
- ▶ **Petit Lemme** : *il existe une solution optimale contenant C_1*

En effet, soit $\mathcal{B} = (C_{i_1}, C_{i_2}, \dots, C_{i_k})$ une solution optimale.

- ▶ Si $C_{i_1} = C_1$, on est content...
- ▶ Sinon, par définition de C_1 , on a $f_1 \leq f_{i_1}$ et $(\mathcal{B} \setminus C_{i_1}) \cup C_1$ est aussi une solution optimale, contenant C_1 cette fois.

Validité de l'algorithme

Théorème

CHOIXCOURSGLOUTON est *optimal* : il renvoie un ensemble maximal (d'indices) de cours compatibles, c-à-d qu'il n'existe pas d'ensemble strictement plus grand de cours compatibles.

Preuve

- ▶ Notons $\mathcal{C} = (C_1, \dots, C_n)$ les cours triés par dates de fin croissantes.
- ▶ **Petit Lemme** : *il existe une solution optimale contenant C_1*

En effet, soit $\mathcal{B} = (C_{i_1}, C_{i_2}, \dots, C_{i_k})$ une solution optimale.

- ▶ Si $C_{i_1} = C_1$, on est content...
 - ▶ Sinon, par définition de C_1 , on a $f_1 \leq f_{i_1}$ et $(\mathcal{B} \setminus C_{i_1}) \cup C_1$ est aussi une solution optimale, contenant C_1 cette fois.
- ▶ \leadsto CHOIXCOURSGLOUTON fait le premier bon choix !

Validité de l'algorithme

Théorème

CHOIXCOURSGLOUTON est *optimal* : il renvoie un ensemble maximal (d'indices) de cours compatibles, c-à-d qu'il n'existe pas d'ensemble strictement plus grand de cours compatibles.

Preuve

- Notons maintenant \mathcal{C}_1 les cours compatibles avec C_1 , triés par dates de fin croissantes (c'est-à-dire les cours dont la date de début est $\geq f_1$). On note $\mathcal{C}_1 = \{C_{j_1}, C_{j_2}, \dots, C_{j_{n_1}}\}$.

Validité de l'algorithme

Théorème

CHOIXCOURSGLOUTON est *optimal* : il renvoie un ensemble maximal (d'indices) de cours compatibles, c-à-d qu'il n'existe pas d'ensemble strictement plus grand de cours compatibles.

Preuve

- ▶ Notons maintenant \mathcal{C}_1 les cours compatibles avec C_1 , triés par dates de fin croissantes (c'est-à-dire les cours dont la date de début est $\geq f_1$). On note $\mathcal{C}_1 = \{C_{j_1}, C_{j_2}, \dots, C_{j_{n_1}}\}$.
- ▶ **Moyen Lemme** : il existe une solution optimale formée de C_1 et d'une solution optimale du problème sur \mathcal{C}_1

En effet, soit $\mathcal{B} = (C_1, C_{i_2}, \dots, C_{i_k})$ une solution optimale du problème contenant C_1 (ça existe par le petit lemme).

- ▶ Si $(C_{i_2}, \dots, C_{i_k})$ n'est pas une solution optimale pour \mathcal{C}_1 , alors il existerait une meilleure solution $(C_{i_2}, \dots, C_{i_{k+1}})$ sur \mathcal{C}_1 .

Validité de l'algorithme

Théorème

CHOIXCOURSGLOUTON est *optimal* : il renvoie un ensemble maximal (d'indices) de cours compatibles, c-à-d qu'il n'existe pas d'ensemble strictement plus grand de cours compatibles.

Preuve

- ▶ Notons maintenant \mathcal{C}_1 les cours compatibles avec C_1 , triés par dates de fin croissantes (c'est-à-dire les cours dont la date de début est $\geq f_1$). On note $\mathcal{C}_1 = \{C_{j_1}, C_{j_2}, \dots, C_{j_{n_1}}\}$.
- ▶ **Moyen Lemme** : il existe une solution optimale formée de C_1 et d'une solution optimale du problème sur \mathcal{C}_1

En effet, soit $\mathcal{B} = (C_1, C_{i_2}, \dots, C_{i_k})$ une solution optimale du problème contenant C_1 (ça existe par le petit lemme).

- ▶ Mais $(C_{i_2}, \dots, C_{i_{k+1}})$ sont tous compatibles avec C_1 , et $(C_1, C_{i_2}, \dots, C_{i_{k+1}})$ serait une meilleure solution que \mathcal{B} au problème de départ, ce qui est exclu !

Validité de l'algorithme

Théorème

CHOIXCOURSGLOUTON est optimal : il renvoie un ensemble maximal (d'indices) de cours compatibles, c-à-d qu'il n'existe pas d'ensemble strictement plus grand de cours compatibles.

Preuve

- ▶ Notons maintenant \mathcal{C}_1 les cours compatibles avec C_1 , triés par dates de fin croissantes (c'est-à-dire les cours dont la date de début est $\geq f_1$). On note $\mathcal{C}_1 = \{C_{j_1}, C_{j_2}, \dots, C_{j_{n_1}}\}$.
- ▶ **Moyen Lemme** : *il existe une solution optimale formée de C_1 et d'une solution optimale du problème sur \mathcal{C}_1*
- ▶ Du coup, par le Petit Lemme, comme il existe une solution optimale de \mathcal{C}_1 commençant par C_{j_1} , il existe une solution optimale du problème de départ commençant par (C_1, C_{j_1}) .
~> CHOIXCOURSGLOUTON fait aussi un second bon choix !

Validité de l'algorithme

Théorème

CHOIXCOURSGLOUTON est optimal : il renvoie un ensemble maximal (d'indices) de cours compatibles, c-à-d qu'il n'existe pas d'ensemble strictement plus grand de cours compatibles.

Preuve

- ▶ On peut mettre en place la récurrence :
 $\mathcal{P}_s =$ 'Si on note $(C_{p_1} (= C_1), C_{p_2}, \dots, C_{p_s})$ les s premiers choix de cours de CHOIXCOURSGLOUTON et \mathcal{C}_s les cours compatibles avec tous ces cours là, alors il existe une solution au problème initial formée de $(C_{p_1}, C_{p_2}, \dots, C_{p_s})$ et d'une solution optimale sur \mathcal{C}_s .'

Validité de l'algorithme

Théorème

CHOIXCOURSGLOUTON est optimal : il renvoie un ensemble maximal (d'indices) de cours compatibles, c-à-d qu'il n'existe pas d'ensemble strictement plus grand de cours compatibles.

Preuve

- ▶ On peut mettre en place la récurrence :
 $\mathcal{P}_s =$ 'Si on note $(C_{p_1}(= C_1), C_{p_2}, \dots, C_{p_s})$ les s premiers choix de cours de CHOIXCOURSGLOUTON et \mathcal{C}_s les cours compatibles avec tous ces cours là, alors il existe une solution au problème initial formée de $(C_{p_1}, C_{p_2}, \dots, C_{p_s})$ et d'une solution optimale sur \mathcal{C}_s .'
- ▶ \mathcal{P}_1 vraie par le Moyen lemme.

Validité de l'algorithme

Théorème

CHOIXCOURSGLOUTON est optimal : il renvoie un ensemble maximal (d'indices) de cours compatibles, c-à-d qu'il n'existe pas d'ensemble strictement plus grand de cours compatibles.

Preuve

- ▶ On peut mettre en place la récurrence :
 $\mathcal{P}_s =$ 'Si on note $(C_{p_1}(= C_1), C_{p_2}, \dots, C_{p_s})$ les s premiers choix de cours de CHOIXCOURSGLOUTON et \mathcal{C}_s les cours compatibles avec tous ces cours là, alors il existe une solution au problème initial formée de $(C_{p_1}, C_{p_2}, \dots, C_{p_s})$ et d'une solution optimale sur \mathcal{C}_s .'
- ▶ Si \mathcal{P}_s vraie, alors par le Moyen Lemme, il existe une solution opt. sur \mathcal{C}_s formée du premier cours $C_{p_{s+1}}$ de \mathcal{C}_s et d'une solution opt. sur les cours de \mathcal{C}_s compatibles avec $C_{p_{s+1}}$. Et \mathcal{P}_{s+1} est vraie !

Validité de l'algorithme

Théorème

CHOIXCOURSGLOUTON est optimal : il renvoie un ensemble maximal (d'indices) de cours compatibles, c-à-d qu'il n'existe pas d'ensemble strictement plus grand de cours compatibles.

Preuve

- ▶ On peut mettre en place la récurrence :
 $\mathcal{P}_s =$ 'Si on note $(C_{p_1}(= C_1), C_{p_2}, \dots, C_{p_s})$ les s premiers choix de cours de CHOIXCOURSGLOUTON et \mathcal{C}_s les cours compatibles avec tous ces cours là, alors il existe une solution au problème initial formée de $(C_{p_1}, C_{p_2}, \dots, C_{p_s})$ et d'une solution optimale sur \mathcal{C}_s .'
- ▶ \rightsquigarrow CHOIXCOURSGLOUTON renvoie une solution optimale au problème !



1. Exemple 1 : choix de cours
2. Qu'est qu'un algorithme glouton ?
3. Exemple 2 : le sac-à-dos (fractionnaire)
4. Exemple spécial : approximation pour SETCOVER dans le plan

Idée générale

Un **algorithme glouton** fait à chaque étape un choix **localement optimal** dans le but d'obtenir à la fin un **optimum global**.

Idée générale

Un **algorithme glouton** fait à chaque étape un choix **localement optimal** dans le but d'obtenir à la fin un **optimum global**.

Exemple du choix de cours

- ▶ Optimum local : cours qui minimise les incompatibilités
- ▶ Optimum global : maximum de cours compatibles

Idée générale

Un **algorithme glouton** fait à chaque étape un choix **localement optimal** dans le but d'obtenir à la fin un **optimum global**.

Exemple du choix de cours

- ▶ Optimum local : cours qui minimise les incompatibilités
- ▶ Optimum global : maximum de cours compatibles

Remarques

- ▶ Construction pas-à-pas d'une solution
 - ▶ Algorithmes simples à concevoir... mais pas toujours parfaits!
- ~> Résolution exacte, approximation, heuristique

Concevoir des algorithmes gloutons

1. Décider d'un **choix glouton**

- ▶ Ajout d'un nouvel élément à la solution en construction
- ▶ Recommencer sur le sous-problème restant

Concevoir des algorithmes gloutons

1. Décider d'un **choix glouton**

- ▶ Ajout d'un nouvel élément à la solution en construction
- ▶ Recommencer sur le sous-problème restant

2. Chercher un cas où **ça ne marche pas**

- ▶ Si on en trouve, retourner en 1.
- ▶ Sinon, continuer en 3.

Concevoir des algorithmes gloutons

Tri par durées croissantes
 $\{[11, 14], [7, 12], [13, 19]\}$

1. Décider d'un **choix glouton**
 - ▶ Ajout d'un nouvel élément à la solution en construction
 - ▶ Recommencer sur le sous-problème restant
2. Chercher un cas où **ça ne marche pas**
 - ▶ Si on en trouve, retourner en 1.
 - ▶ Sinon, continuer en 3.

Concevoir des algorithmes gloutons

Tri par durées croissantes
 $\{[11, 14], [7, 12], [13, 19]\}$

1. Décider d'un **choix glouton**
 - ▶ Ajout d'un nouvel élément à la solution en construction
 - ▶ Recommencer sur le sous-problème restant
2. Chercher un cas où **ça ne marche pas**
 - ▶ Si on en trouve, retourner en 1.
 - ▶ Sinon, continuer en 3.
3. **Démontrer** que l'algorithme est correct
 - ▶ Il existe une solution optimale contenant le choix local
 - ▶ Choix local + glouton pour le reste \rightsquigarrow solution optimale

Concevoir des algorithmes gloutons

Tri par durées croissantes
 $\{[11, 14], [7, 12], [13, 19]\}$

1. Décider d'un **choix glouton**
 - ▶ Ajout d'un nouvel élément à la solution en construction
 - ▶ Recommencer sur le sous-problème restant
2. Chercher un cas où **ça ne marche pas**
 - ▶ Si on en trouve, retourner en 1.
 - ▶ Sinon, continuer en 3.
3. **Démontrer** que l'algorithme est correct
 - ▶ Il existe une solution optimale contenant le choix local
 - ▶ Choix local + glouton pour le reste \rightsquigarrow solution optimale
4. Étudier la **complexité** de l'algorithme

Algorithme glouton générique

Problème générique

Entrée Un ensemble fini X , avec une valeur v_x pour tout $x \in X$
Un ensemble \mathcal{I} de solutions acceptables $A \subset X$

Hyp. Une sous-solution reste acceptable : si $A \in \mathcal{I}$ et $B \subset A$, $B \in \mathcal{I}$

Sortie Une solution acceptable $A \in \mathcal{I}$ qui maximise/minimise
$$v_A = \sum_{x \in A} v_x$$

Algorithme glouton générique

Problème générique

Entrée Un ensemble fini X , avec une valeur v_x pour tout $x \in X$
Un ensemble \mathcal{I} de solutions acceptables $A \subset X$

Hyp. Une sous-solution reste acceptable : si $A \in \mathcal{I}$ et $B \subset A$, $B \in \mathcal{I}$

Sortie Une solution acceptable $A \in \mathcal{I}$ qui maximise/minimise
$$v_A = \sum_{x \in A} v_x$$

Algorithme : GLOUTONGÉNÉRIQUE(X, \mathcal{I})

Trier X par *valeurs* décroissantes/croissantes + critère glouton de choix ;

$S \leftarrow \emptyset$;

pour $x \in X$ (*dans l'ordre du tri*) **faire**

si $S \cup \{x\} \in \mathcal{I}$ **alors** $S \leftarrow S \cup \{x\}$;

retourner S

Théorème des algorithmes gloutons

Théorème

Si pour toute entrée (X, \mathcal{I}) , il existe une solution optimale S tq

- ▶ le premier élément x_0 de X appartient à S*
- ▶ $S \setminus \{x_0\}$ soit une solution optimale de $(X \setminus \{x_0\}, \mathcal{I}')$ où $\mathcal{I}' = \{A \setminus \{x_0\} : A \in \mathcal{I}, x_0 \in A\}$*

Alors GLOUTONGÉNÉRIQUE est optimal.

Théorème des algorithmes gloutons

Théorème

Si pour toute entrée (X, \mathcal{I}) , il existe une solution optimale S tq

- ▶ *le premier élément x_0 de X appartient à S*
- ▶ *$S \setminus \{x_0\}$ soit une solution optimale de $(X \setminus \{x_0\}, \mathcal{I}')$ où $\mathcal{I}' = \{A \setminus \{x_0\} : A \in \mathcal{I}, x_0 \in A\}$*

Alors GLOUTONGÉNÉRIQUE est optimal.

Exemple du choix de cours

- ▶ X : ensemble des cours, avec $v_x = 1$ pour tout x
- ▶ \mathcal{I} : ensembles de cours compatibles
- ▶ Tri : dates de fin croissantes
- ▶ Preuve :
 - ▶ Il existe un ensemble de cours optimal contenant le 1^{er} cours
 - ▶ En enlevant le 1^{er} cours, il reste un ensemble optimal pour les cours commençant après la fin du 1^{er} cours

Théorème des algorithmes gloutons

Théorème

Si pour toute entrée (X, \mathcal{I}) , il existe une solution optimale S tq

- ▶ *le premier élément x_0 de X appartient à S*
- ▶ *$S \setminus \{x_0\}$ soit une solution optimale de $(X \setminus \{x_0\}, \mathcal{I}')$ où $\mathcal{I}' = \{A \setminus \{x_0\} : A \in \mathcal{I}, x_0 \in A\}$*

Alors GLOUTONGÉNÉRIQUE est optimal.

Preuve par récurrence sur $|X|$

- ▶ Si $|X| = 0$, la solution optimale est \emptyset
- ▶ Soit (X, \mathcal{I}) une entrée avec $|X| > 0$. Par hyp. de récurrence, GLOUTONGÉNÉRIQUE trouve une solution optimale S' pour $(X \setminus \{x_0\}, \mathcal{I}')$. Donc $S' \cup \{x_0\}$ est optimale pour (X, \mathcal{I}) .

En pratique

- ▶ Il existe une théorie générale des algorithmes gloutons
 - ▶ basée sur la notion de **matroïde**
 - ▶ mais certains algorithmes « type glouton » ne rentrent pas exactement dans le moule

En pratique

- ▶ Il existe une théorie générale des algorithmes gloutons
 - ▶ basée sur la notion de **matroïde**
 - ▶ mais certains algorithmes « type glouton » ne rentrent pas exactement dans le moule
- ▶ Dans ce cours : étude de plusieurs exemples
 - ▶ utilisation du théorème pour faciliter les preuves

En pratique

- ▶ Il existe une théorie générale des algorithmes gloutons
 - ▶ basée sur la notion de **matroïde**
 - ▶ mais certains algorithmes « type glouton » ne rentrent pas exactement dans le moule
- ▶ Dans ce cours : étude de plusieurs exemples
 - ▶ utilisation du théorème pour faciliter les preuves

Objectifs :

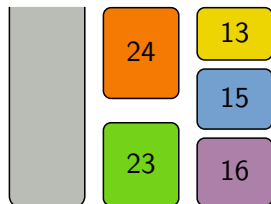
- ▶ Savoir tenter une stratégie gloutonne
- ▶ Savoir détecter si elle marche ou non
- ▶ Savoir l'analyser (validité et complexité)

1. Exemple 1 : choix de cours
2. Qu'est qu'un algorithme glouton ?
3. Exemple 2 : le sac-à-dos (fractionnaire)
4. Exemple spécial : approximation pour SETCOVER dans le plan

Problème du sac-à-dos

Définition du problème

- Entrée** un ensemble d'objets, ayant une taille t_i et une valeur v_i
une taille T de sac-à-dos
- Sortie** un sous-ensemble des objets qui rentrent dans le sac
($\sum_i t_i \leq T$) et qui maximise la valeur totale ($V = \sum_i v_i$)

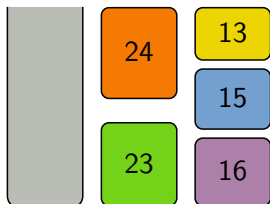


Problème du sac-à-dos

Définition du problème

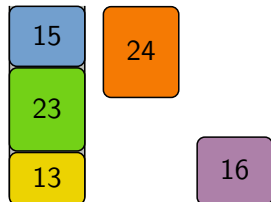
Entrée un ensemble d'objets, ayant une taille t_i et une valeur v_i
une taille T de sac-à-dos

Sortie un sous-ensemble des objets qui rentrent dans le sac
($\sum_i t_i \leq T$) et qui maximise la valeur totale ($V = \sum_i v_i$)



- ▶ Problème célèbre car utile
 - ▶ en théorie
 - ▶ en pratique
 - ▶ en cryptographie
- ▶ Difficile (NP-difficile \rightsquigarrow Bloc 4)

Problème du sac-à-dos



Définition du problème

Entrée un ensemble d'objets, ayant une taille t_i et une valeur v_i
une taille T de sac-à-dos

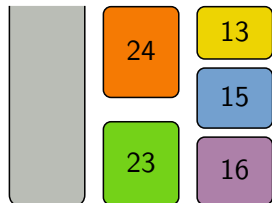
Sortie un sous-ensemble des objets qui rentrent dans le sac
($\sum_i t_i \leq T$) et qui maximise la valeur totale ($V = \sum_i v_i$)

- ▶ Problème célèbre car utile
 - ▶ en théorie
 - ▶ en pratique
 - ▶ en cryptographie
- ▶ Difficile (NP-difficile \rightsquigarrow Bloc 4)

Problème du sac-à-dos fractionnaire

Objets *fractionnables* :

on peut n'en prendre qu'une partie



Définition

Entrée un ensemble d'objets, ayant une taille t_i et une valeur v_i
une taille T de sac-à-dos

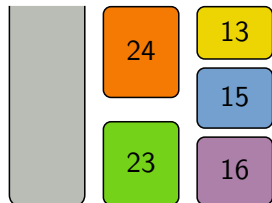
Sortie une fraction $x_i \in [0, 1]$ pour chaque objet, telle que

- ▶ le total ne dépasse pas la taille du sac : $\sum_i x_i t_i \leq T$
- ▶ la valeur totale est maximale : $V = \sum_i x_i v_i$

Problème du sac-à-dos fractionnaire

Objets *fractionnables* :

on peut n'en prendre qu'une partie



Définition

Entrée un ensemble d'objets, ayant une taille t_i et une valeur v_i
une taille T de sac-à-dos

Sortie une fraction $x_i \in [0, 1]$ pour chaque objet, telle que

- ▶ le total ne dépasse pas la taille du sac : $\sum_i x_i t_i \leq T$
- ▶ la valeur totale est maximale : $V = \sum_i x_i v_i$

- ▶ Problème simplifié !
- ▶ Approche pour résoudre le sac-à-dos

Problème du sac-à-dos fractionnaire

Objets *fractionnables* :
on peut n'en prendre qu'une partie

Définition

Entrée un ensemble d'objets, ayant une taille t_i et une valeur v_i
une taille T de sac-à-dos

Sortie une fraction $x_i \in [0, 1]$ pour chaque objet, telle que

- ▶ le total ne dépasse pas la taille du sac : $\sum_i x_i t_i \leq T$
- ▶ la valeur totale est maximale : $V = \sum_i x_i v_i$

- ▶ Problème simplifié !
- ▶ Approche pour résoudre le sac-à-dos



Algorithme glouton

Choix glouton : choisir l'objet de meilleur *rapport qualité - prix*

Algorithme glouton

Choix glouton : choisir l'objet de meilleur *rapport qualité - prix*

Algorithme : SÀDFRACGLOUTON(O, T)

Trier les objets $O_i = (t_i, v_i)$ par v_i/t_i **décroissant**

$R \leftarrow T$ // Reste libre dans le sac-à-dos

pour $i = 1$ à n (dans l'ordre du tri) **faire**

si $t_i \leq R$ **alors**

$x_i \leftarrow 1$

$R \leftarrow R - t_i$

sinon

$x_i \leftarrow R/t_i$

$R \leftarrow 0$

retourner (x_1, \dots, x_n)

Algorithme glouton

Choix glouton : choisir l'objet de meilleur *rapport qualité - prix*

Algorithme : SÀDFRACGLOUTON(O, T)

Trier les objets $O_i = (t_i, v_i)$ par v_i/t_i **décroissant**

$R \leftarrow T$ // Reste libre dans le sac-à-dos

pour $i = 1$ à n (dans l'ordre du tri) **faire**

si $t_i \leq R$ **alors**

$x_i \leftarrow 1$

$R \leftarrow R - t_i$

sinon

$x_i \leftarrow R/t_i$

$R \leftarrow 0$

retourner (x_1, \dots, x_n)

Lemme

La complexité de SÀDFRACGLOUTON est $O(n \log n)$.

Validité de l'algorithme

Lemme

Soit $O = \{(t_1, v_1), \dots, (t_n, v_n)\}$ un ensemble d'objets et T une taille de sac-à-dos, où $v_1/t_1 \geq v_2/t_2 \geq \dots \geq v_n/t_n$. Alors il existe une solution optimale (x_1, \dots, x_n) sur l'entrée (O, T) telle que

- ▶ $x_1 = \begin{cases} 1 & \text{si } t_1 \leq T \\ T/t_1 & \text{sinon} \end{cases}$
- ▶ (x_2, \dots, x_n) est solution optimale sur l'entrée $\{(t_2, v_2), \dots, (t_n, v_n)\}$ et $T - t_1$

Preuve : voir TD...

↪ Optimalité de SÀDFRACGLOUTON d'après le théorème des algorithmes gloutons !

1. Exemple 1 : choix de cours
2. Qu'est qu'un algorithme glouton ?
3. Exemple 2 : le sac-à-dos (fractionnaire)
4. Exemple spécial : approximation pour SETCOVER dans le plan

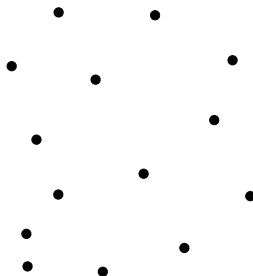
Le problème

SETCOVER

Entrée n maisons placées dans le plan

Sortie un ensemble minimal de maisons où placer une antenne Wifi :

- ▶ chaque antenne a une portée de 500 m
- ▶ toutes les maisons doivent être couvertes



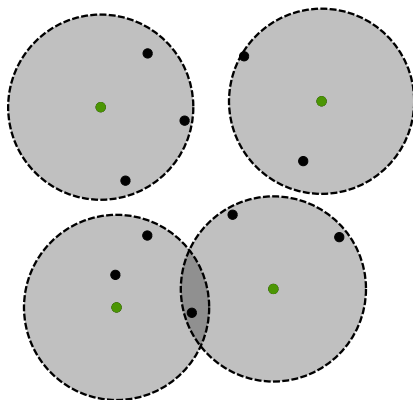
Le problème

SETCOVER

Entrée n maisons placées dans le plan

Sortie un ensemble minimal de maisons où placer une antenne Wifi :

- ▶ chaque antenne a une portée de 500 m
- ▶ toutes les maisons doivent être couvertes



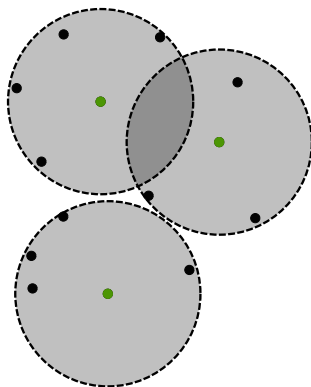
Le problème

SETCOVER

Entrée n maisons placées dans le plan

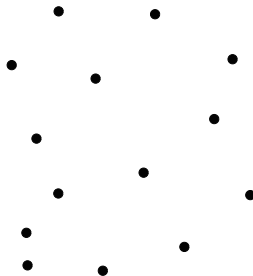
Sortie un ensemble minimal de maisons où placer une antenne Wifi :

- ▶ chaque antenne a une portée de 500 m
- ▶ toutes les maisons doivent être couvertes



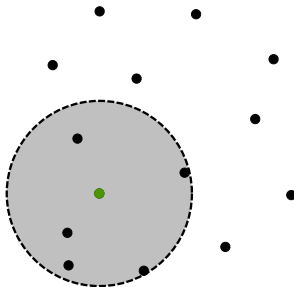
Choix glouton

À chaque étape, on choisit la maison qui permet de couvrir le plus de maisons non encore couvertes



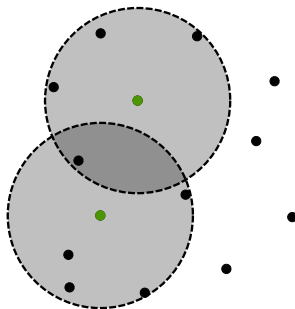
Choix glouton

À chaque étape, on choisit la maison qui permet de couvrir le plus de maisons non encore couvertes



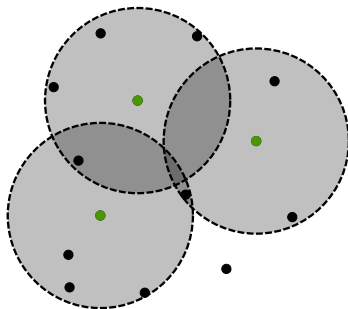
Choix glouton

À chaque étape, on choisit la maison qui permet de couvrir le plus de maisons non encore couvertes



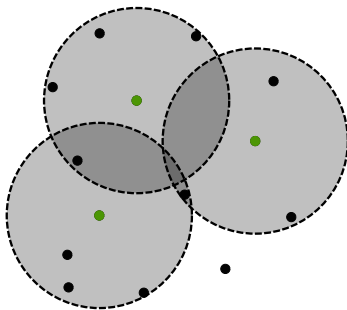
Choix glouton

À chaque étape, on choisit la maison qui permet de couvrir le plus de maisons non encore couvertes



Choix glouton

À chaque étape, on choisit la maison qui permet de couvrir le plus de maisons non encore couvertes



Choix non optimal mais...

Propriétés du choix glouton

Lemme

L'algorithme présenté peut être implanté en temps $O(n^3)$

Preuve : exercice de TD !

Propriétés du choix glouton

Lemme

L'algorithme présenté peut être implanté en temps $O(n^3)$

Preuve : exercice de TD !

Lemme

Si k est le nombre minimal d'antennes nécessaires, l'algorithme trouve toujours une solution avec $\leq k \cdot \log n$ antennes

Propriétés du choix glouton

Lemme

L'algorithme présenté peut être implanté en temps $O(n^3)$

Preuve : exercice de TD !

Lemme

Si k est le nombre minimal d'antennes nécessaires, l'algorithme trouve toujours une solution avec $\leq k \cdot \log n$ antennes

Preuve

- Notons n_t le nbr de maisons *non-couvertes* après l'étape t .
- $n_0 = n$

Propriétés du choix glouton

Lemme

L'algorithme présenté peut être implanté en temps $O(n^3)$

Preuve : exercice de TD !

Lemme

Si k est le nombre minimal d'antennes nécessaires, l'algorithme trouve toujours une solution avec $\leq k \cdot \log n$ antennes

Preuve

- ▶ Notons n_t le nbr de maisons *non-couvertes* après l'étape t .
 - ▶ $n_0 = n$
 - ▶ Dans la solution optimale, k antennes couvrent toutes les maisons, et en particulier les n_t maisons non-couvertes après l'étape t
 - ▶ Il existe donc un emplacement qui va permettre de couvrir au moins $\frac{n_t}{k}$ maisons non-couvertes.

Propriétés du choix glouton

Lemme

L'algorithme présenté peut être implanté en temps $O(n^3)$

Preuve : exercice de TD !

Lemme

Si k est le nombre minimal d'antennes nécessaires, l'algorithme trouve toujours une solution avec $\leq k \cdot \log n$ antennes

Preuve

- ▶ Notons n_t le nbr de maisons *non-couvertes* après l'étape t .
 - ▶ $n_0 = n$
 - ▶ Dans la solution optimale, k antennes couvrent toutes les maisons, et en particulier les n_t maisons non-couvertes après l'étape t
 - ▶ Il existe donc un emplacement qui va permettre de couvrir au moins $\frac{n_t}{k}$ maisons non-couvertes.
 - ▶ On a : $n_{t+1} \leq n_t - \frac{n_t}{k} = (1 - \frac{1}{k}) \cdot n_t$

Propriétés du choix glouton

Lemme

L'algorithme présenté peut être implanté en temps $O(n^3)$

Preuve : exercice de TD !

Lemme

Si k est le nombre minimal d'antennes nécessaires, l'algorithme trouve toujours une solution avec $\leq k \cdot \log n$ antennes

Preuve

- ▶ Notons n_t le nbr de maisons *non-couvertes* après l'étape t .
- ▶ En résumé : $n_0 = n$ et pour tout $t \geq 0$ on a $n_{t+1} \leq (1 - \frac{1}{k}) \cdot n_t$

Propriétés du choix glouton

Lemme

L'algorithme présenté peut être implanté en temps $O(n^3)$

Preuve : exercice de TD !

Lemme

Si k est le nombre minimal d'antennes nécessaires, l'algorithme trouve toujours une solution avec $\leq k \cdot \log n$ antennes

Preuve

- ▶ Notons n_t le nbr de maisons *non-couvertes* après l'étape t .
- ▶ En résumé : $n_0 = n$ et pour tout $t \geq 0$ on a $n_{t+1} \leq (1 - \frac{1}{k}) \cdot n_t$
- ▶ Comme pour $x > 0$ on a $1 - x \leq 2^{-x}$, on obtient
$$n_{t+1} \leq 2^{-\frac{1}{k}} \cdot n_t$$

Propriétés du choix glouton

Lemme

L'algorithme présenté peut être implanté en temps $O(n^3)$

Preuve : exercice de TD !

Lemme

Si k est le nombre minimal d'antennes nécessaires, l'algorithme trouve toujours une solution avec $\leq k \cdot \log n$ antennes

Preuve

- ▶ Notons n_t le nbr de maisons *non-couvertes* après l'étape t .
- ▶ En résumé : $n_0 = n$ et pour tout $t \geq 0$ on a $n_{t+1} \leq (1 - \frac{1}{k}) \cdot n_t$
- ▶ Comme pour $x > 0$ on a $1 - x \leq 2^{-x}$, on obtient

$$n_{t+1} \leq 2^{-\frac{1}{k}} \cdot n_t$$

- ▶ D'où, $n_t \leq 2^{-\frac{1}{k}} \cdot n_{t-1} \leq 2^{-\frac{2}{k}} \cdot n_{t-2} \leq \dots \leq 2^{-\frac{t}{k}} \cdot n_0 = 2^{-\frac{t}{k}} \cdot n$

Propriétés du choix glouton

Lemme

L'algorithme présenté peut être implanté en temps $O(n^3)$

Preuve : exercice de TD !

Lemme

Si k est le nombre minimal d'antennes nécessaires, l'algorithme trouve toujours une solution avec $\leq k \cdot \log n$ antennes

Preuve

- ▶ Notons n_t le nbr de maisons *non-couvertes* après l'étape t .
- ▶ En résumé : $n_0 = n$ et pour tout $t \geq 0$ on a $n_{t+1} \leq (1 - \frac{1}{k}) \cdot n_t$
- ▶ Comme pour $x > 0$ on a $1 - x \leq 2^{-x}$, on obtient

$$n_{t+1} \leq 2^{-\frac{1}{k}} \cdot n_t$$

- ▶ D'où, $n_t \leq 2^{-\frac{1}{k}} \cdot n_{t-1} \leq 2^{-\frac{2}{k}} \cdot n_{t-2} \leq \dots \leq 2^{-\frac{t}{k}} \cdot n_0 = 2^{-\frac{t}{k}} \cdot n$
- ▶ On est sûr d'avoir couvert toutes les maisons dès que $2^{-\frac{t}{k}} \cdot n < 1$, c-à-d, $t > k \cdot \log n$

Conclusion

Bilan

Pourquoi des algorithmes gloutons ?

- ▶ Algorithmes souvent simples et rapides...
- ▶ ... parfois optimaux
- ▶ ... parfois avec de bonnes propriétés
- ▶ ... parfois qui marchent en pratique
- ▶ ... parfois parfaitement inutiles !

Bilan

Pourquoi des algorithmes gloutons ?

- ▶ Algorithmes souvent simples et rapides...
- ▶ ... parfois optimaux
- ▶ ... parfois avec de bonnes propriétés
- ▶ ... parfois qui marchent en pratique
- ▶ ... parfois parfaitement inutiles !

Comment les utiliser ?

1. Chercher un choix glouton
2. Démontrer que c'est un bon choix (en **théorie** ou pratique)
3. Étudier la complexité obtenue