Chapitre 4
Algorithmes glouton

HLIN401 : Algorithmique et Complexité

Université de Montpellier
2018 – 2019

1. Premier exemple : choix de cours

2. Qu'est qu'un algorithme glouton ?

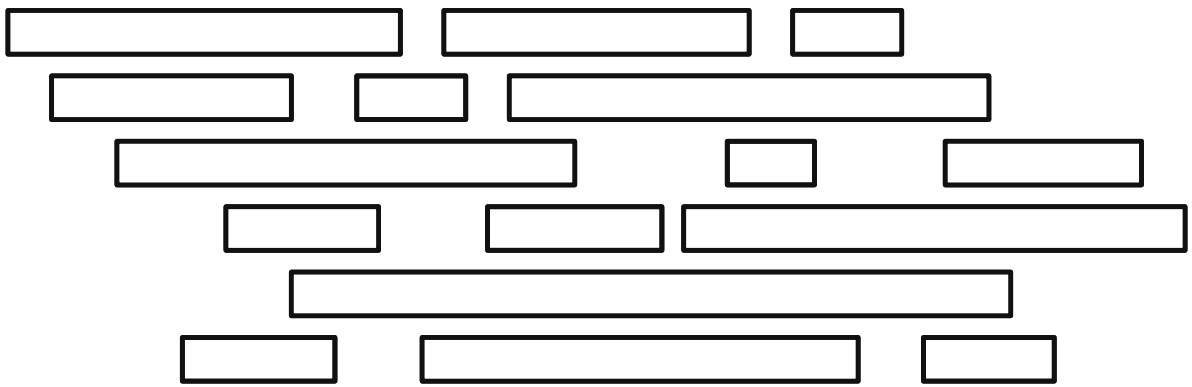
3. Exemple 2 : le sac-à-dos (fractionnaire)

4. Exemple spécial : approximation pour SetCover dans le plan

Définition du problème

Entrée un ensemble C de cours $C_i = (d_i, f_i)$ [début, fin]

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

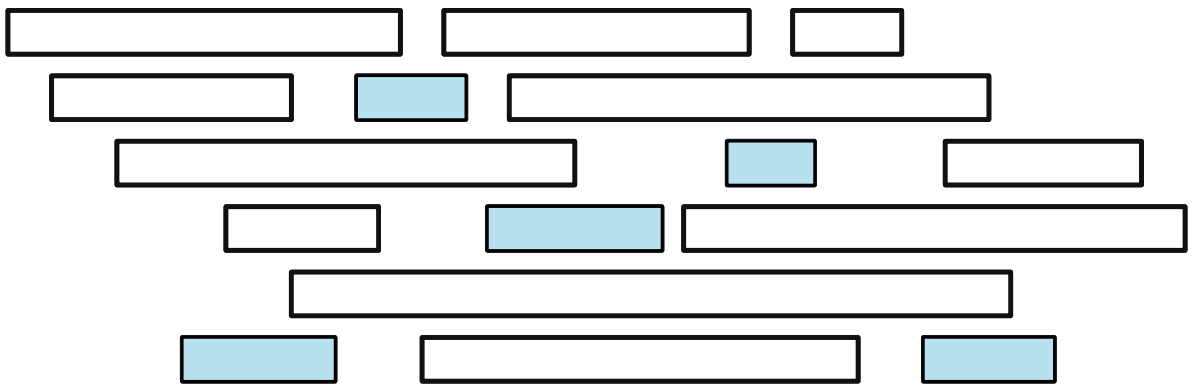


modifié d'après *Algorithms* de J. Erickson

Définition du problème

Entrée un ensemble C de cours $C_i = (d_i, f_i)$ [début, fin]

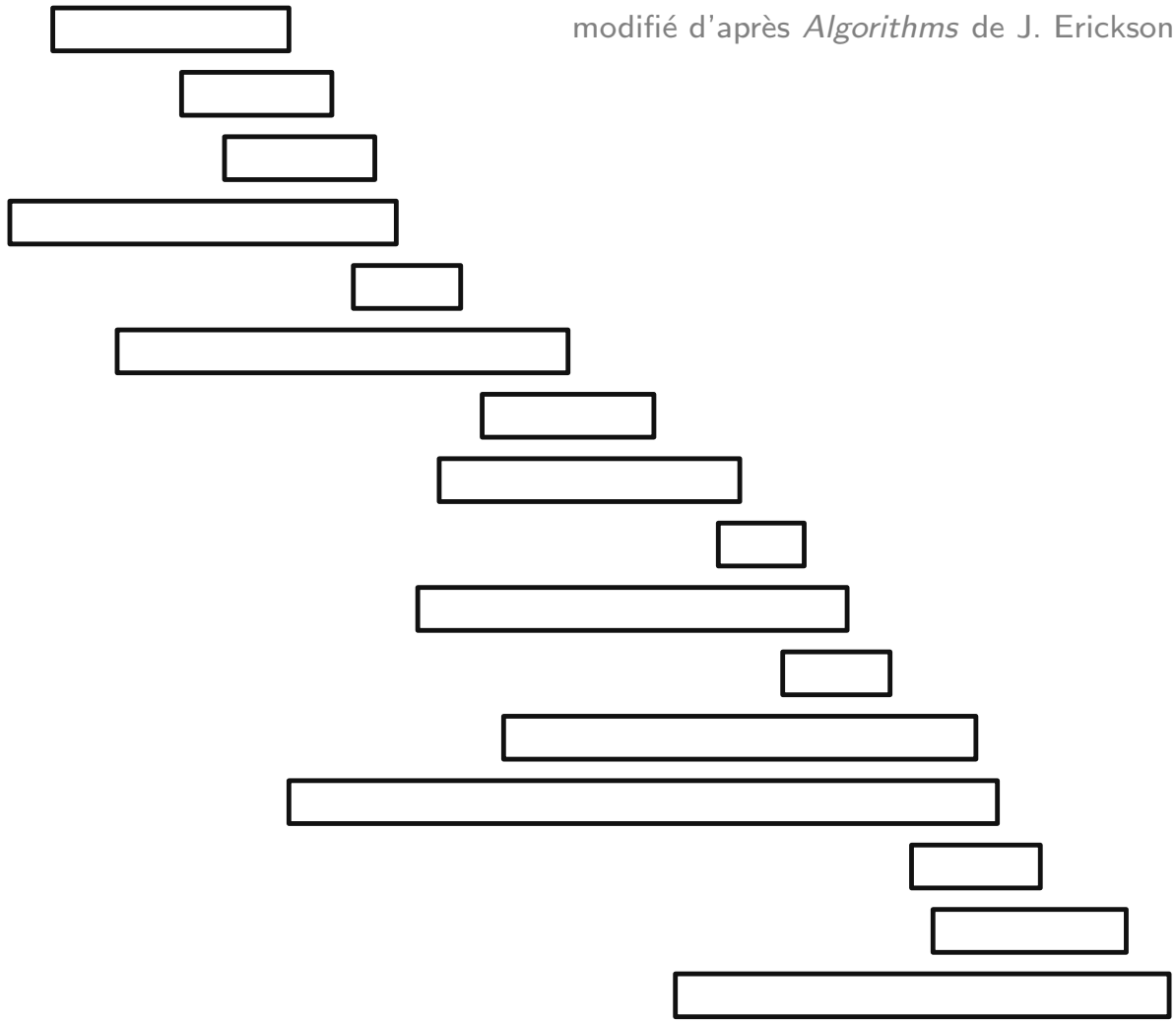
Sortie un ensemble ordonné maximal de cours $(C_{i_1}, \ldots, C_{i_k})$ tels que pour tout $j < k$, $f_{i_j} \leq d_{i_{j+1}}$ $\rightsquigarrow$ cours compatibles



modifié d'après *Algorithms* de J. Erickson

Idée gloutonne

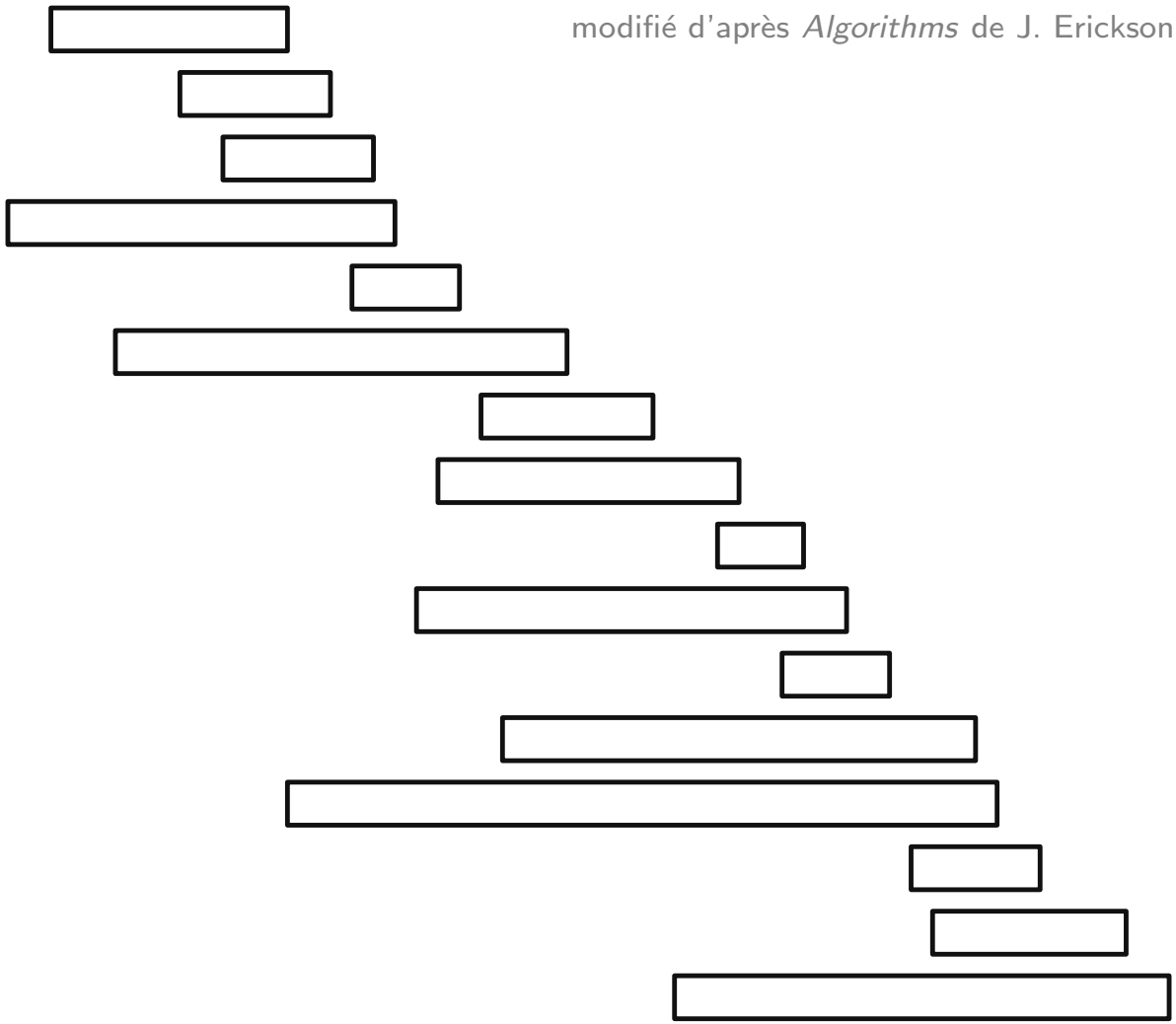
- Tri des cours par **dates de fin croissantes**



modifié d'après *Algorithms* de J. Erickson

Idée gloutonne

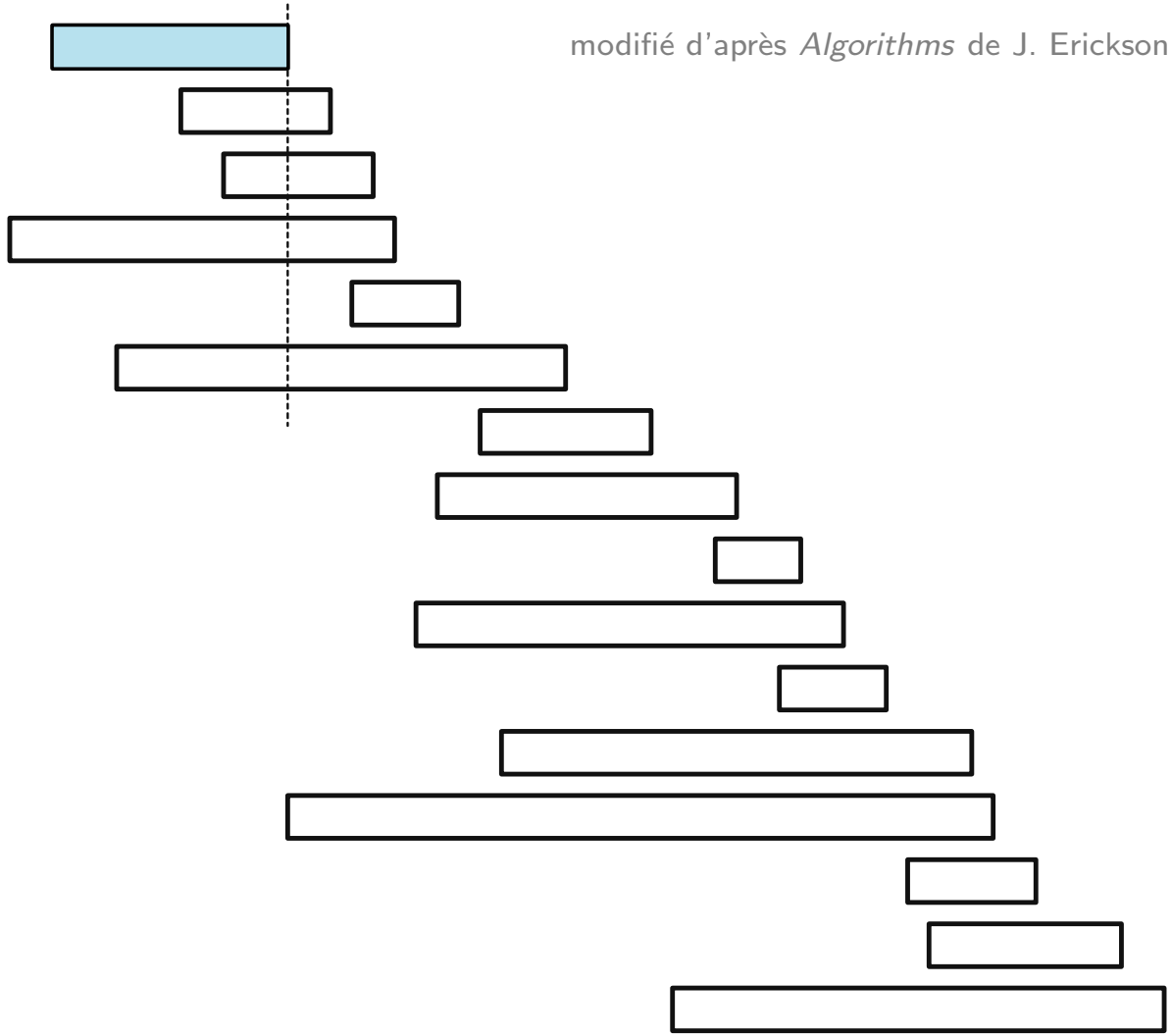
- Tri des cours par **dates de fin croissantes**
- Choix *glouton* : sélectionner le cours qui finit **le plus tôt**



modifié d'après *Algorithms* de J. Erickson

Idée gloutonne

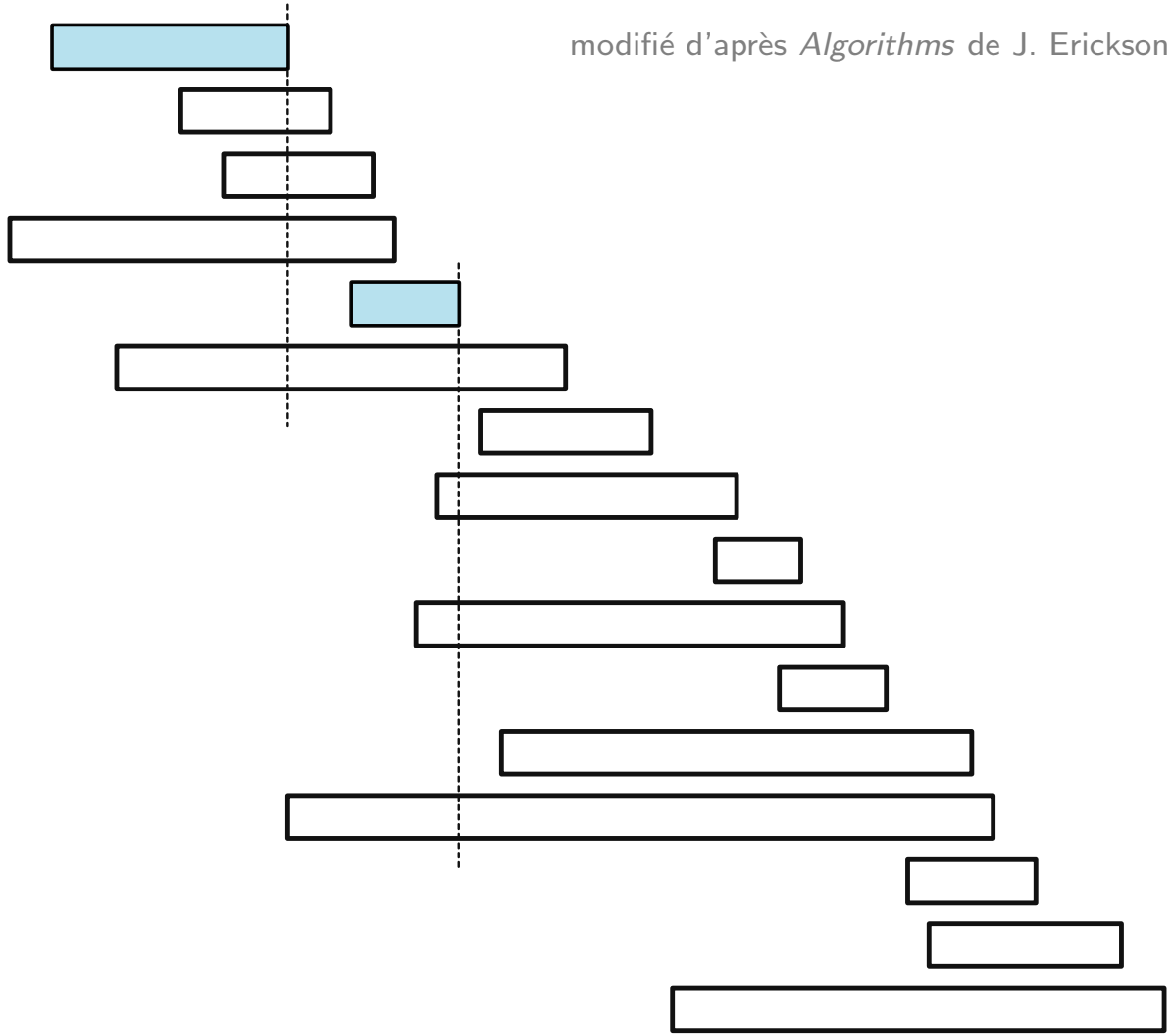
- Tri des cours par **dates de fin croissantes**
- Choix *glouton* : sélectionner le cours qui finit **le plus tôt**



modifié d'après *Algorithms* de J. Erickson

Idée gloutonne

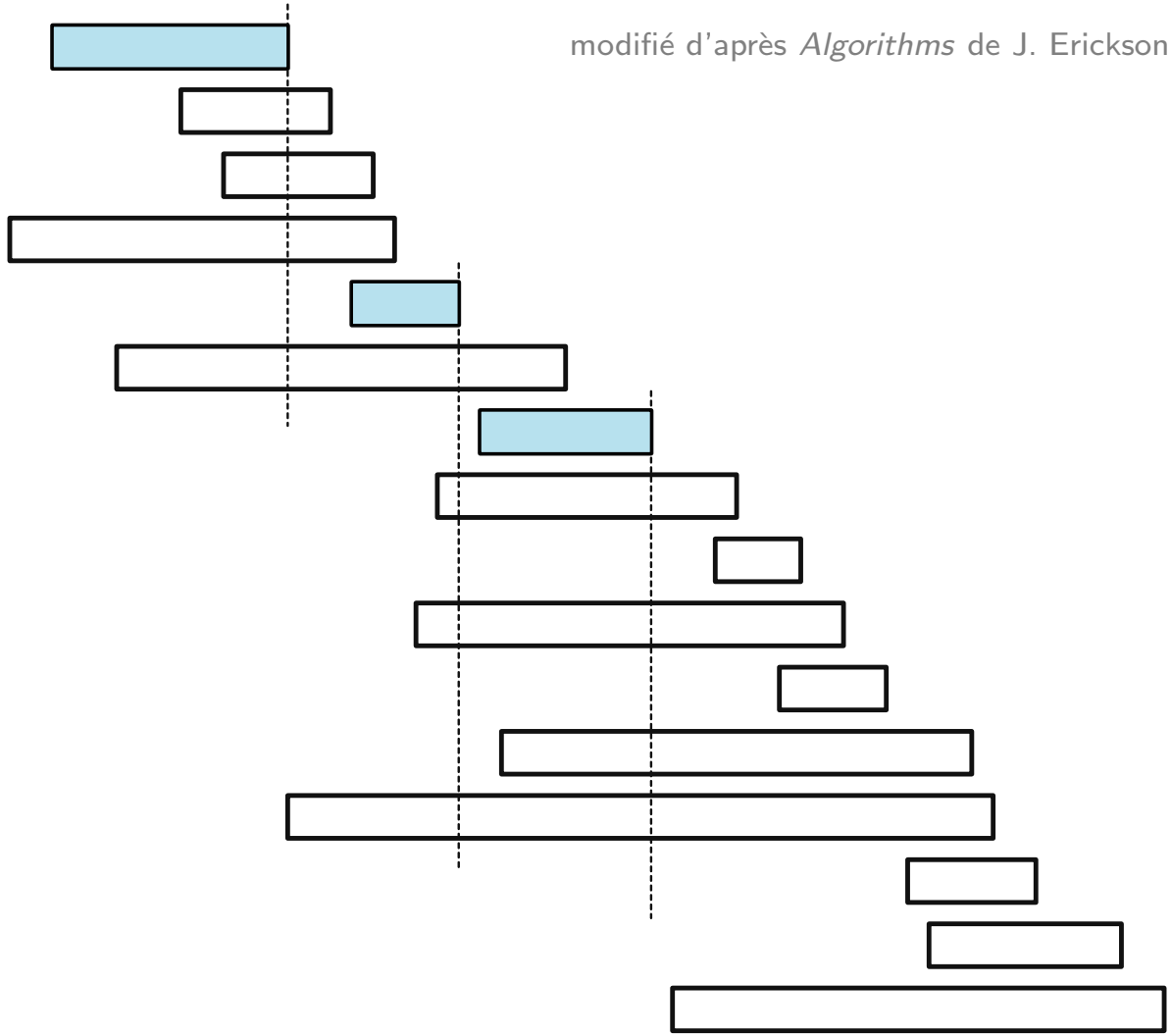
- Tri des cours par **dates de fin croissantes**
- Choix *glouton* : sélectionner le cours qui finit **le plus tôt**



modifié d'après *Algorithms* de J. Erickson

Idée gloutonne

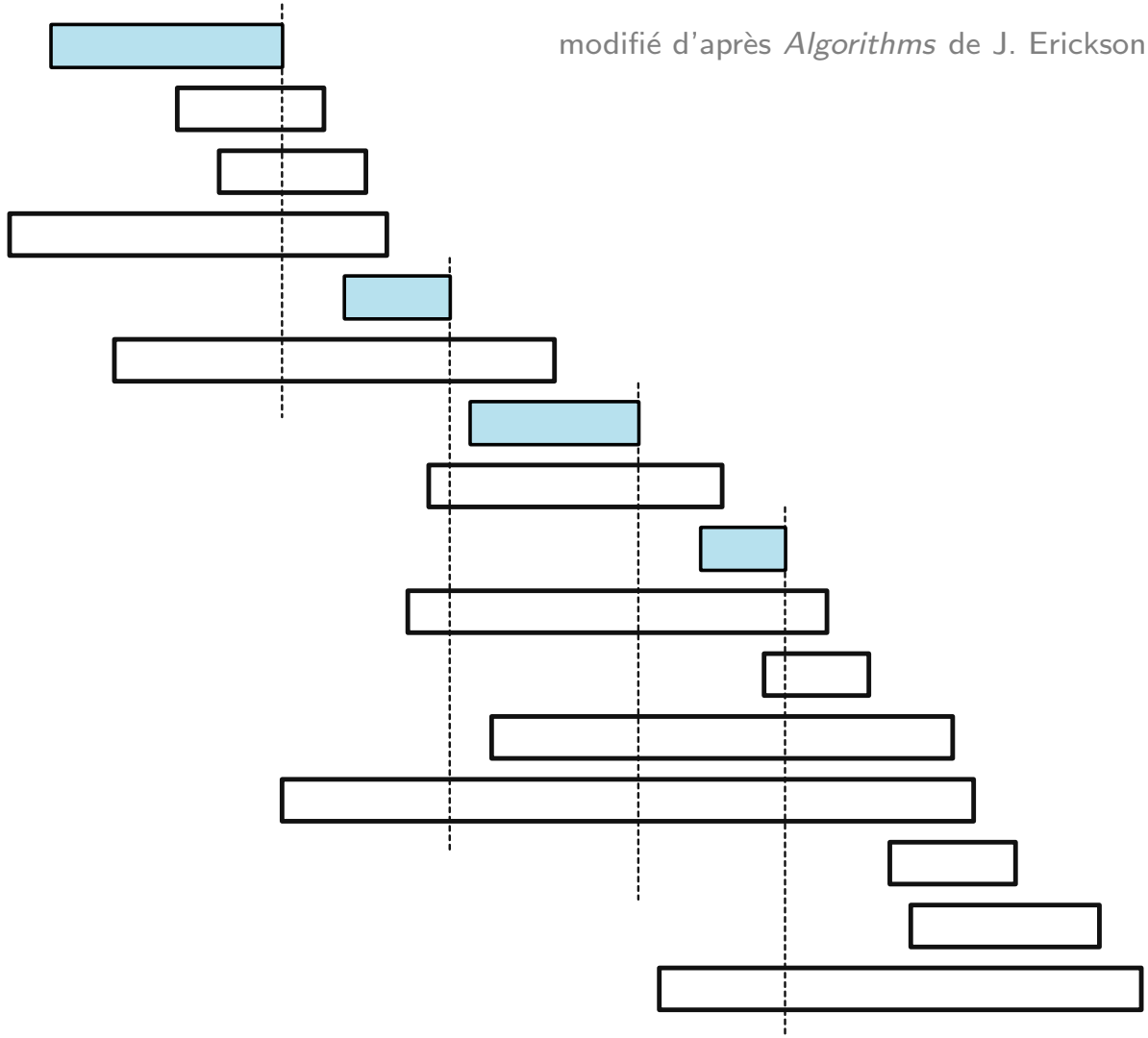
- Tri des cours par **dates de fin croissantes**
- Choix *glouton* : sélectionner le cours qui finit **le plus tôt**



modifié d'après *Algorithms* de J. Erickson

Idée gloutonne

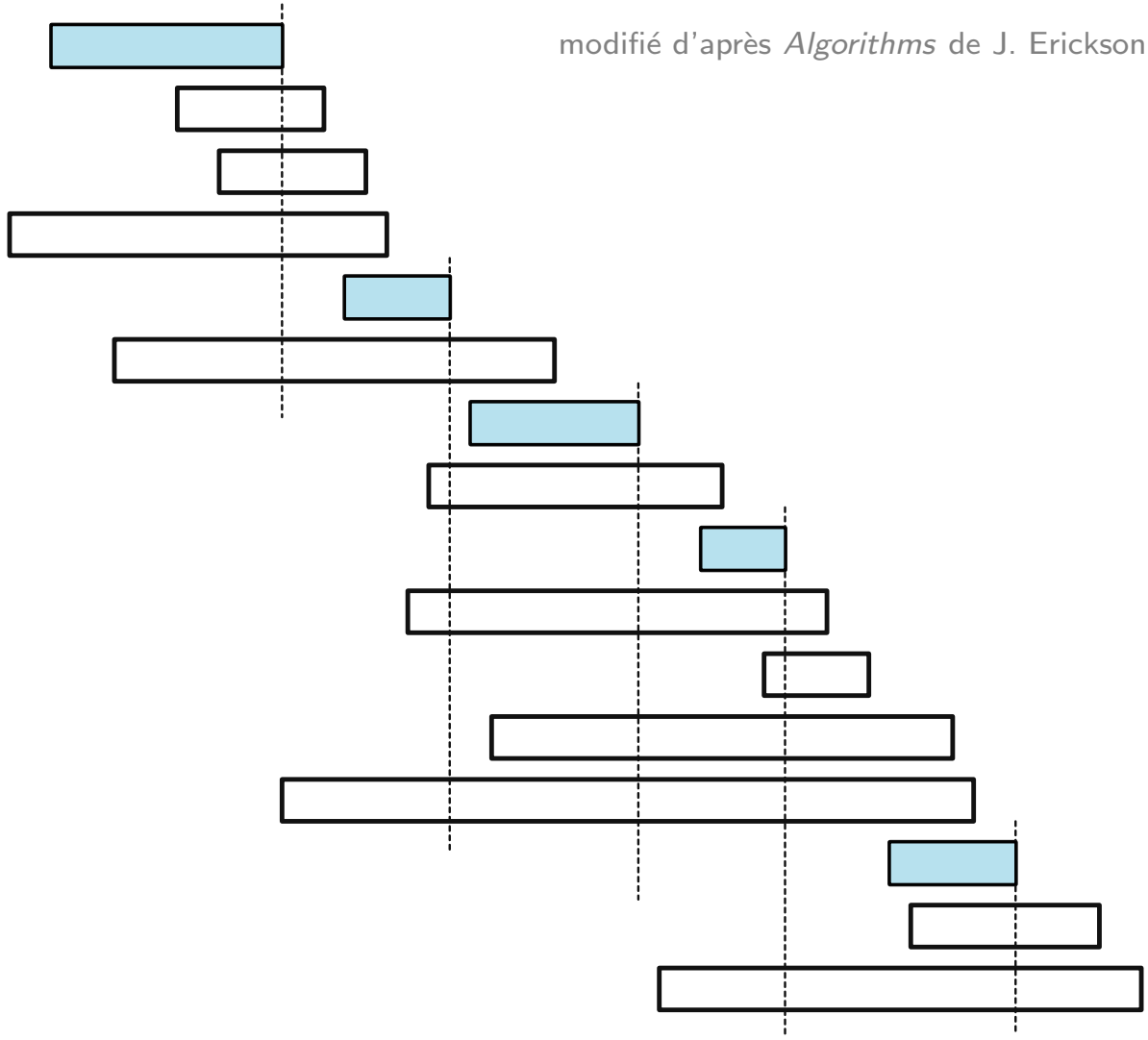
- Tri des cours par **dates de fin croissantes**
- Choix *glouton* : sélectionner le cours qui finit **le plus tôt**



modifié d'après *Algorithms* de J. Erickson

Idée gloutonne

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



modifié d'après *Algorithms* de J. Erickson

Algorithme glouton

Algorithme : ChoixCoursGlouton(C)

Trier C en fonction des fins
$I \leftarrow \{1\}$ // Indices des cours choisis
$f \leftarrow$ Fin($C[1]$) // Fin du dernier cours choisi
pour $i = 2$ *à* n **faire**
    si Début($C[i]$) $\geq f$ **alors**
        $I \leftarrow I \cup \{i\}$
        $f \leftarrow$ Fin($C[i]$)
retourner I

Algorithme glouton

Algorithme : ChoixCoursGlouton(C)

Trier C en fonction des fins
$I \leftarrow \{1\}$ // Indices des cours choisis
$f \leftarrow$ Fin($C[1]$) // Fin du dernier cours choisi
pour $i = 2$ *à* n **faire**
    si Début($C[i]$) $\geq f$ **alors**
        $I \leftarrow I \cup \{i\}$
        $f \leftarrow$ Fin($C[i]$)
retourner I

Question

Quelle est la complexité de ChoixCoursGlouton *?*

Algorithme glouton

Algorithme : ChoixCoursGlouton(C)

Trier C en fonction des fins
$I \leftarrow \{1\}$ // Indices des cours choisis
$f \leftarrow$ Fin($C[1]$) // Fin du dernier cours choisi
pour $i = 2$ *à* n **faire**
    si Début($C[i]$) $\geq f$ **alors**
        $I \leftarrow I \cup \{i\}$
        $f \leftarrow$ Fin($C[i]$)
retourner I

Question

Quelle est la complexité de ChoixCoursGlouton *?* $\rightsquigarrow O(n \log n)$

Validité de l'algorithme

Théorème

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

Preuve au tableau

1. Premier exemple : 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 !

$\rightsquigarrow$ 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
$\{(8, 11), (0, 9), (10, 19)\}$

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
$\{(8, 11), (0, 9), (10, 19)\}$

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
$\{(8, 11), (0, 9), (10, 19)\}$

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 $v_A = \sum_{x \in A} v_x$

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 $v_A = \sum_{x \in A} v_x$

Algorithme : GloutonGénérique$(X, \mathcal{I})$

Trier X par *valeurs* croissantes (+ éventuellement autre critère)
$S \leftarrow \emptyset$
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 appartienne 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 appartienne 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 appartienne 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

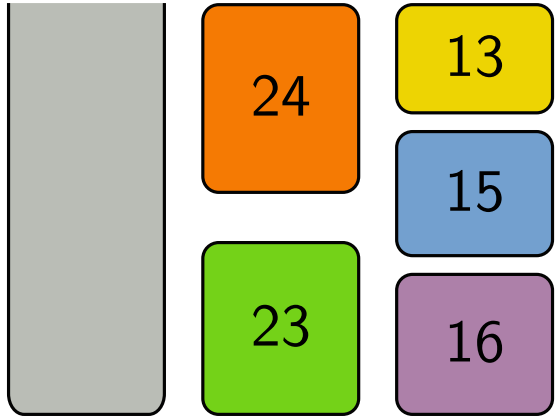
En pratique

- 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)

Problème du sac-à-dos

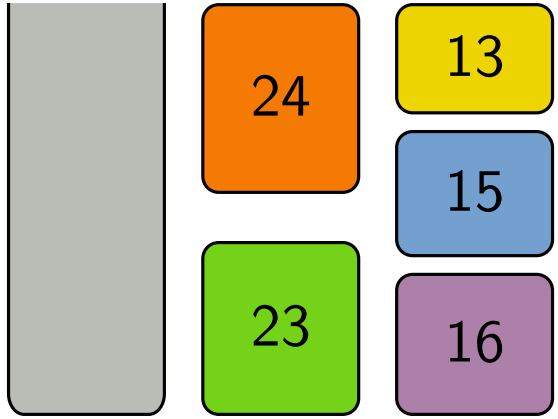


Définition du problème

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



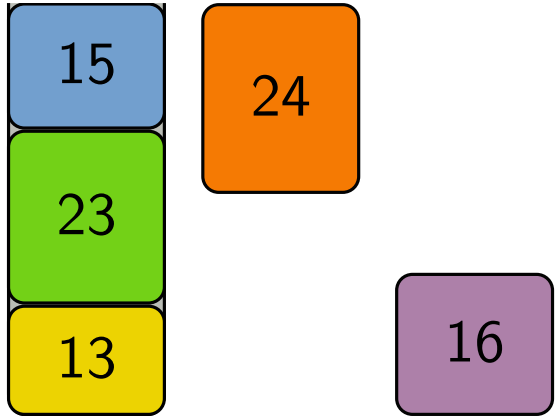
Définition du problème

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-complet $\rightsquigarrow$ HLIN612)

Problème du sac-à-dos



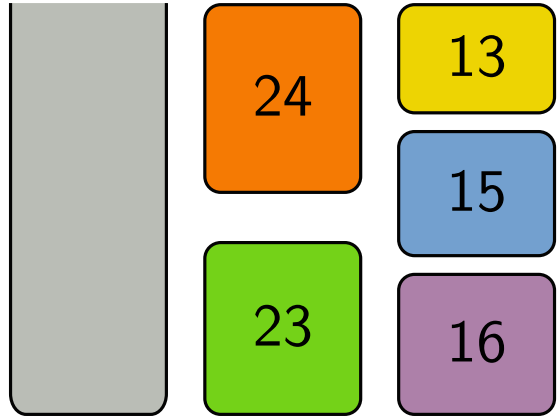
Définition du problème

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-complet $\rightsquigarrow$ HLIN612)

Problème du sac-à-dos fractionnaire



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

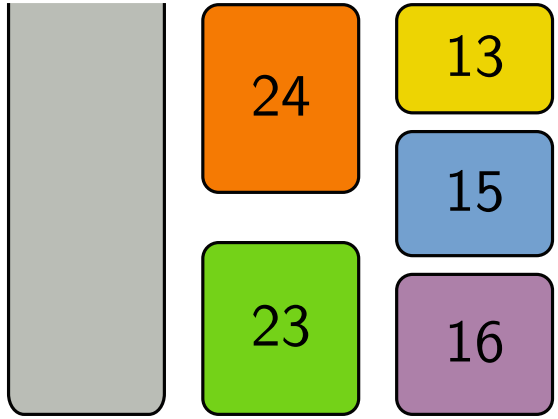
Définition

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

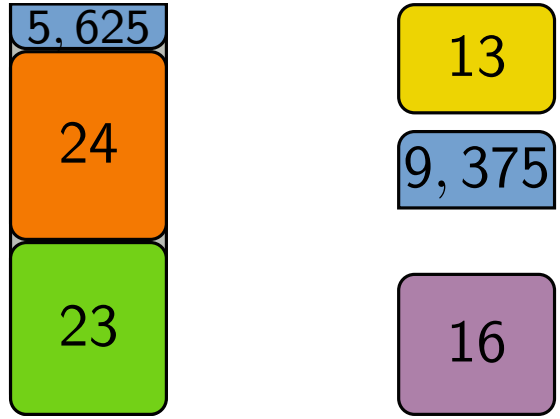
Entrée un ensemble d'objets, ayant une taille t_i et une valeur v_i
une taille T de sac-à-dos

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 $(x_1, \ldots, x_n)$

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 $(x_1, \ldots, x_n)$

Lemme

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

Validité de l'algorithme

Lemme

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

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

Preuve du lemme au tableau

$\rightsquigarrow$ Optimalité de SàdFracGlouton d'après le théorème des algorithmes gloutons !

1. Premier exemple : 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

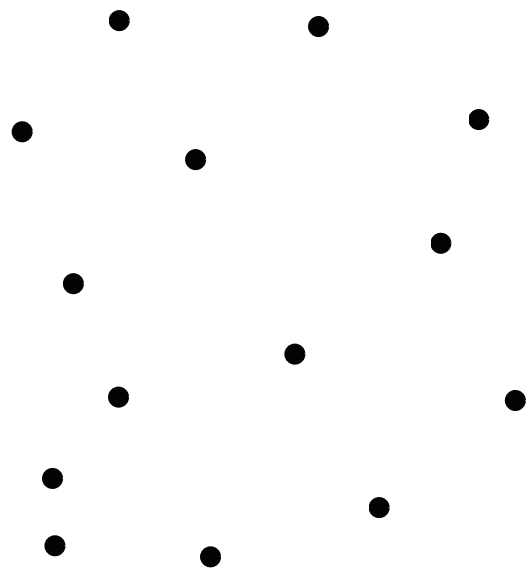
Le problème

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



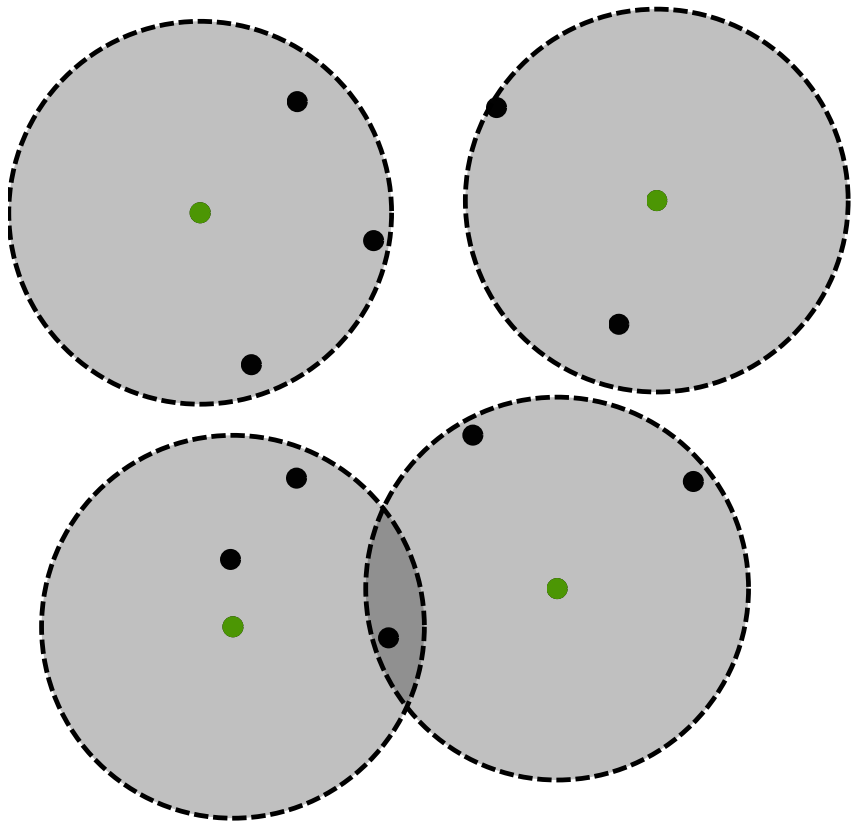
Le problème

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



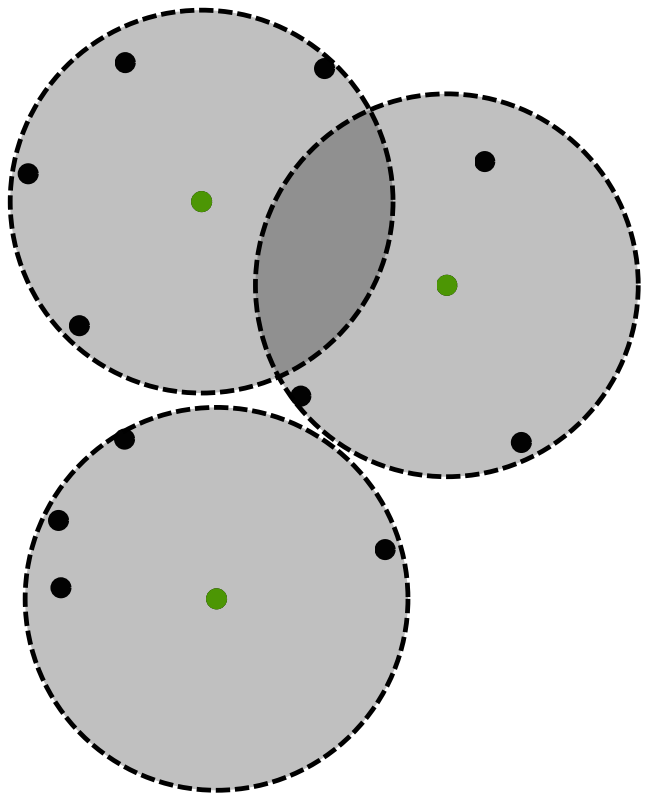
Le problème

SetCover

Entrée n maisons placées dans le plan

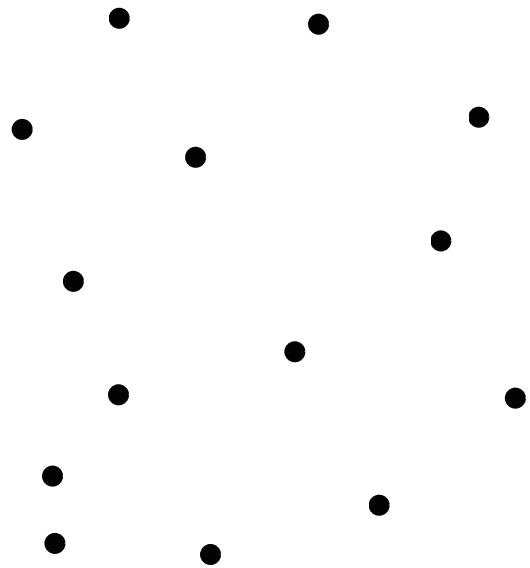
Sortie un ensemble minimal de maisons où placer une antenne Wifi :

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



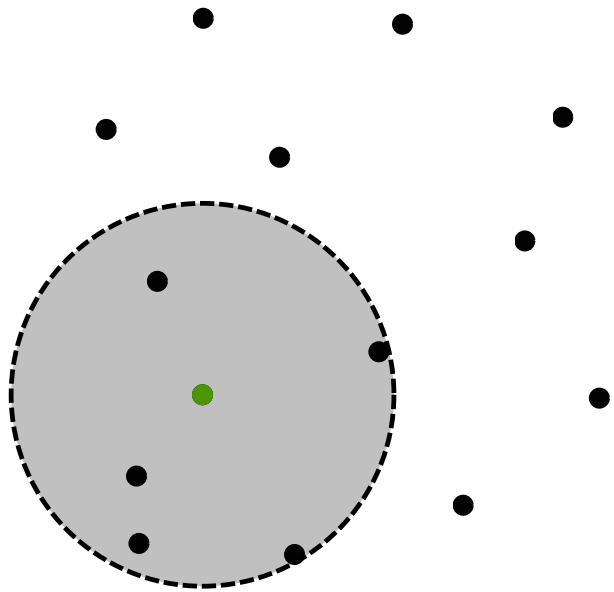
Choix glouton

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



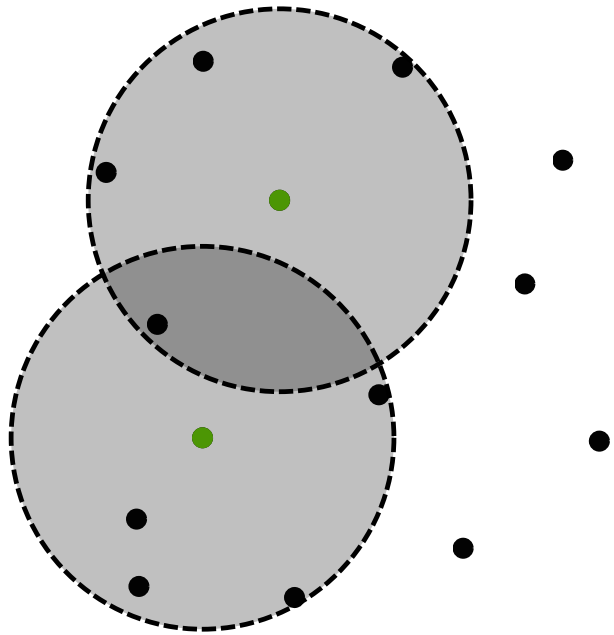
Choix glouton

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



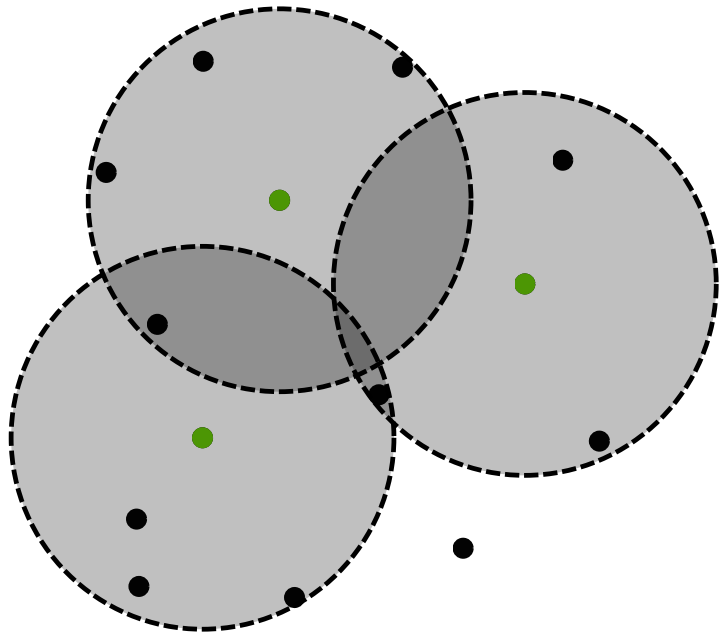
Choix glouton

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



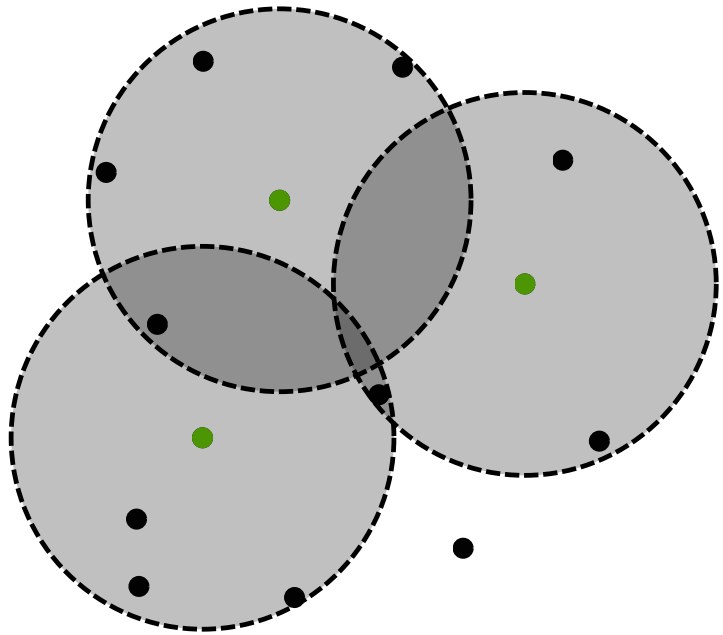
Choix glouton

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 en TD

Propriétés du choix glouton

Lemme

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

Preuve en TD

Lemme

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

Preuve au tableau

Propriétés du choix glouton

Lemme

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

Preuve en TD

Lemme

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

Preuve au tableau

Remarque

On ne connaît pas d'algorithme polynomial qui fasse mieux... et il est (très) probable qu'il n'en existe pas !

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