M1 Info - Optimisation et Recherche Opérationnelle

Cours 6 - Schema d'approximation en temps polynomial

Sac a dos

Semestre Automne 2021-2022 - Université Claude Bernard Lyon 1

Christophe Crespelle
christophe.crespelle@univ-lyon1.fr

département
Informatique

Faculté des Sciences et Technologies
Université Claude Bernard Lyon 1

Probleme du sac a dos

- **Entrée :** un poids limite $W \in \mathbb{R}$ et n objets qui ont une valeur $v_i \in \mathbb{R}$ et un poids $w_i \in \mathbb{R}$

Probleme du sac a dos

- **Entrée :** un poids limite $W \in \mathbb{R}$ et n objets qui ont une valeur $v_i \in \mathbb{R}$ et un poids $w_i \in \mathbb{R}$
- **Sortie :** un sous ensemble $S \subseteq [\![1, n]\!]$ d'objets tel que $\sum_{i \in S} w_i \leq W$ et $\sum_{i \in S} v_i$ est maximum

Probleme du sac a dos

- **Entrée :** un poids limite $W \in \mathbb{R}$ et n objets qui ont une valeur $v_i \in \mathbb{R}$ et un poids $w_i \in \mathbb{R}$

- **Sortie :** un sous ensemble $S \subseteq [\![1, n]\!]$ d'objets tel que $\sum_{i \in S} w_i \leq W$ et $\sum_{i \in S} v_i$ est maximum

Exemple :

v_i	11	7	18	13	9
w_i	1	4	2	5	3
	obj1	obj2	obj3	obj4	obj5

$S_3 = \{obj_1, obj_3, obj_4\}$ $w(S_3) = 11+18+13 = 42$

$w(S_3) = 1+2+5 = 8 \leq 8$

$W = 8$

$S_1 = \{obj_4, obj_5\}$ $w(S_1) \leq 8$ et $w(S_1) = 22$

$S_2 = \{obj_1, obj_2, obj_3\}$ $w(S_2) = 1+4+2 = 7 \leq 8$ $w(S_2) = 11+7+18 = 36$

Probleme du sac a dos

- **Entrée :** un poids limite $W \in \mathbb{R}$ et n objets qui ont une valeur $v_i \in \mathbb{R}$ et un poids $w_i \in \mathbb{R}$

- **Sortie :** un sous ensemble $S \subseteq [\![1, n]\!]$ d'objets tel que $\sum_{i \in S} w_i \leq W$ et $\sum_{i \in S} v_i$ est maximum

Difficulte de calcul : **NP-complet**

$\exists$? sac $v(\text{sac}) \geq h$

(avec contrainte $w(\text{sac}) \leq W$)

Probleme du sac a dos

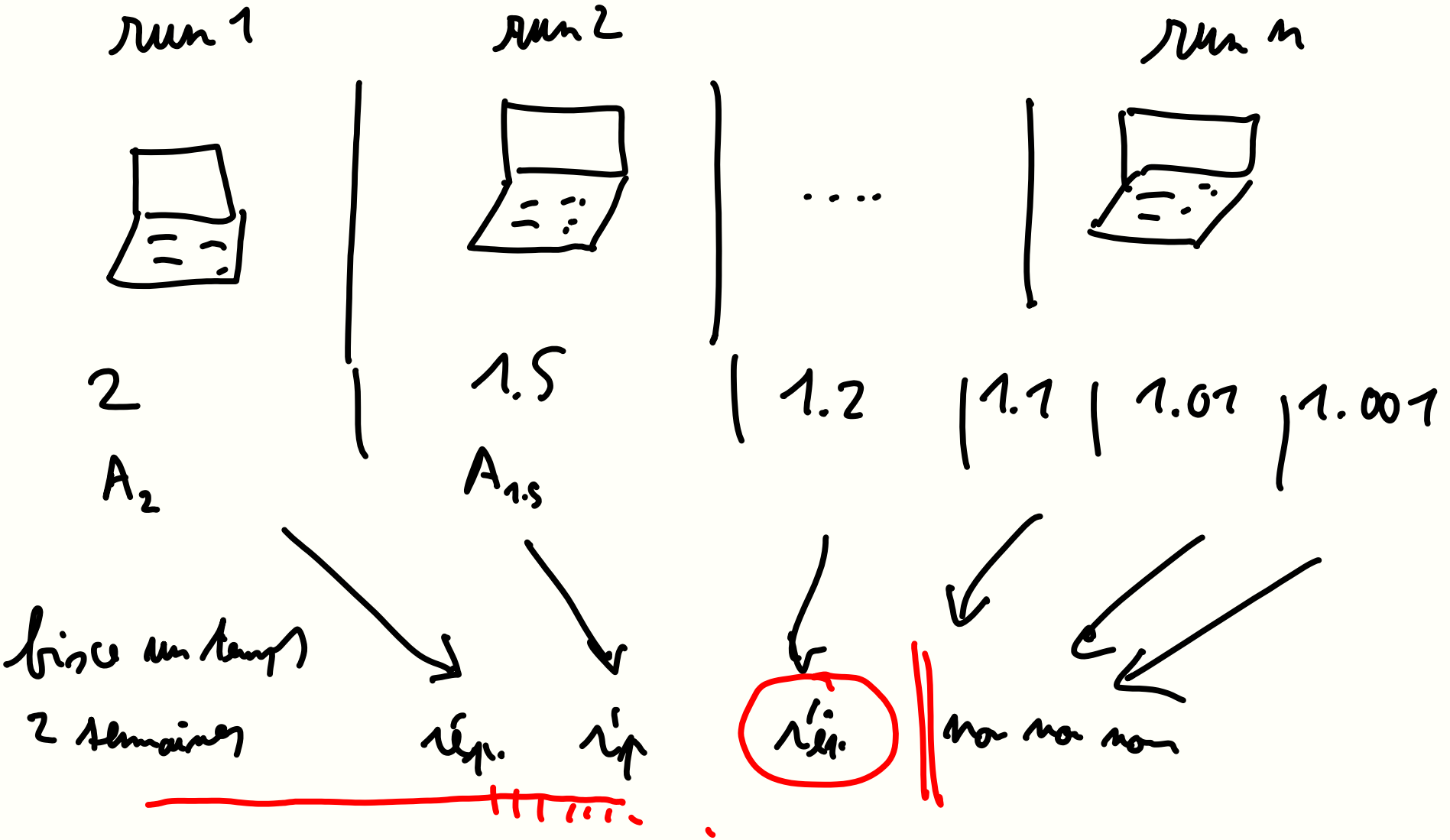
- **Entrée :** un poids limite $W \in \mathbb{R}$ et n objets qui ont une valeur $v_i \in \mathbb{R}$ et un poids $w_i \in \mathbb{R}$

- **Sortie :** un sous ensemble $S \subseteq [\![1, n]\!]$ d'objets tel que $\sum_{i \in S} w_i \leq W$ et $\sum_{i \in S} v_i$ est maximum

Difficulte de calcul : **NP-complet**

On va construire un schema polynomial d'approximation, c'est a dire un algorithme A_ϵ qui donne une approximation aussi bonne qu'on veut (ratio $1 + \epsilon$, pour n'importe quel $\epsilon > 0$)

Schema polynomial d'approximation



Algo exact (exponentiel) pour sac a dos a valeurs entieres

- Les valeurs v_i des objets sont des entiers (les poids sont reels)
- Le probleme reste **NP-complet**

Algo exact (exponentiel) pour sac a dos a valeurs entieres

- Les valeurs v_i des objets sont des entiers (les poids sont reels)
- Le probleme reste **NP-complet**
- L'algo resoud le probleme de maniere exacte et dans une complexite $O(n^2 v^*)$, avec $v* = \max_{i=1}^{n}\{v_i\}$.

Algo exact (exponentiel) pour sac a dos a valeurs entieres

- Les valeurs v_i des objets sont des entiers (les poids sont reels)
- Le probleme reste **NP-complet**
- L'algo resoud le probleme de maniere exacte et dans une complexite $O(n^2 v^*)$, avec $v* = \max_{i=1}^{n}\{v_i\}$.

L'approche : programmation dynamique

Définition

$\overline{OPT}(i, V)$ est le poids minimum W d'un sac a dos de valeur au moins V qui n'utilise que les objets $\{1, 2, \ldots, i\}$.

Algo exact (exponentiel) pour sac a dos a valeurs entieres

- Les valeurs v_i des objets sont des entiers (les poids sont reels)
- Le probleme reste **NP-complet**
- L'algo resoud le probleme de maniere exacte et dans une complexite $O(n^2 v^*)$, avec $v* = \max_{i=1}^{n}\{v_i\}$.

L'approche : programmation dynamique

Définition

$\overline{OPT}(i, V)$ est le poids minimum W d'un sac a dos de valeur au moins V qui n'utilise que les objets $\{1, 2, \ldots, i\}$.

Formule de recurrence :

- si $V > \sum_{j=1}^{i-1} v_j$, alors $\overline{OPT}(i, V) = w_i + \overline{OPT}(i-1, \max\{0, V - v_i\})$
- sinon, $\overline{OPT}(i, V) = \min\{\overline{OPT}(i-1, V), w_i + \overline{OPT}(i-1, \max\{0, V - v_i\})\}$

La formule de recurrence

Principe : discussion sur prend on l'objet i dans $\overline{OPT}(i, V)$?

La formule de recurrence

Principe : discussion sur prend on l'objet i dans $\overline{OPT}(i, V)$?

- si $V > \sum_{j=1}^{i-1} v_j$

La formule de recurrence

Principe : discussion sur prend on l'objet i dans $\overline{OPT}(i, V)$?

- si $V > \sum_{j=1}^{i-1} v_j$ alors on est oblige de prendre l'objet i

La formule de recurrence

Principe : discussion sur prend on l'objet i dans $\overline{OPT}(i, V)$?

- si $V > \sum_{j=1}^{i-1} v_j$ alors on est oblige de prendre l'objet i
  - si $v_i \geq V$

La formule de recurrence

Principe : discussion sur prend on l'objet i dans $\overline{OPT}(i, V)$?

- si $V > \sum_{j=1}^{i-1} v_j$ alors on est oblige de prendre l'objet i
  - si $v_i \geq V$ alors $\overline{OPT}(i, V) = w_i$

La formule de recurrence

Principe : discussion sur prend on l'objet i dans $\overline{OPT}(i, V)$?

- si $V > \sum_{j=1}^{i-1} v_j$ alors on est oblige de prendre l'objet i
  - si $v_i \geq V$ alors $\overline{OPT}(i, V) = w_i$
  - si $v_i < V$

$w_i + \overline{OPT}(i-1, V-v_i)$

La formule de recurrence

Principe : discussion sur prend on l'objet i dans $\overline{OPT}(i, V)$?

- si $V > \sum_{j=1}^{i-1} v_j$ alors on est oblige de prendre l'objet i
  - si $v_i \geq V$ alors $\overline{OPT}(i, V) = w_i$
  - si $v_i < V$ alors $\overline{OPT}(i, V) = w_i + \overline{OPT}(i-1, V - v_i)$

La formule de recurrence

Principe : discussion sur prend on l'objet i dans $\overline{OPT}(i, V)$?

- si $V > \sum_{j=1}^{i-1} v_j$ alors on est oblige de prendre l'objet i
  - si $v_i \geq V$ alors $\overline{OPT}(i, V) = w_i$
  - si $v_i < V$ alors $\overline{OPT}(i, V) = w_i + \overline{OPT}(i-1, V - v_i)$

  Dans tous les cas :
  $\overline{OPT}(i, V) = w_i + \overline{OPT}(i-1, \max\{0, V - v_i\})$

$S = \{\}$ $\quad \overline{OPT}(-, 0) = 0$

La formule de recurrence

Principe : discussion sur prend on l'objet i dans $\overline{OPT}(i, V)$?

- si $V > \sum_{j=1}^{i-1} v_j$ alors on est oblige de prendre l'objet i
  - si $v_i \geq V$ alors $\overline{OPT}(i, V) = w_i$
  - si $v_i < V$ alors $\overline{OPT}(i, V) = w_i + \overline{OPT}(i-1, V - v_i)$

  Dans tous les cas :
  $\overline{OPT}(i, V) = w_i + \overline{OPT}(i-1, \max\{0, V - v_i\})$
- si $V \leq \sum_{j=1}^{i-1} v_j$

La formule de recurrence

Principe : discussion sur prend on l'objet i dans $\overline{OPT}(i, V)$?

- si $V > \sum_{j=1}^{i-1} v_j$ alors on est oblige de prendre l'objet i
  - si $v_i \geq V$ alors $\overline{OPT}(i, V) = w_i$
  - si $v_i < V$ alors $\overline{OPT}(i, V) = w_i + \overline{OPT}(i-1, V - v_i)$

  Dans tous les cas :
  $\overline{OPT}(i, V) = w_i + \overline{OPT}(i-1, \max\{0, V - v_i\})$
- si $V \leq \sum_{j=1}^{i-1} v_j$
  - soit on prend l'objet i

La formule de recurrence

Principe : discussion sur prend on l'objet i dans $\overline{OPT}(i, V)$?

- si $V > \sum_{j=1}^{i-1} v_j$ alors on est oblige de prendre l'objet i
  - si $v_i \geq V$ alors $\overline{OPT}(i, V) = w_i$
  - si $v_i < V$ alors $\overline{OPT}(i, V) = w_i + \overline{OPT}(i-1, V - v_i)$

  Dans tous les cas :
  $\overline{OPT}(i, V) = w_i + \overline{OPT}(i-1, \max\{0, V - v_i\})$
- si $V \leq \sum_{j=1}^{i-1} v_j$
  - soit on prend l'objet i et alors
    $\overline{OPT}(i, V) = w_i + \overline{OPT}(i-1, \max\{0, V - v_i\})$

La formule de recurrence

Principe : discussion sur prend on l'objet i dans $\overline{OPT}(i, V)$?

- si $V > \sum_{j=1}^{i-1} v_j$ alors on est oblige de prendre l'objet i
  - si $v_i \geq V$ alors $\overline{OPT}(i, V) = w_i$
  - si $v_i < V$ alors $\overline{OPT}(i, V) = w_i + \overline{OPT}(i-1, V - v_i)$

  Dans tous les cas :
  $\overline{OPT}(i, V) = w_i + \overline{OPT}(i-1, \max\{0, V - v_i\})$
- si $V \leq \sum_{j=1}^{i-1} v_j$
  - soit on prend l'objet i et alors
    $\overline{OPT}(i, V) = w_i + \overline{OPT}(i-1, \max\{0, V - v_i\})$
  - soit on ne le prend pas

$\overline{OPT}(i-1, V)$

La formule de recurrence

Principe : discussion sur prend on l'objet i dans $\overline{OPT}(i, V)$?

- si $V > \sum_{j=1}^{i-1} v_j$ alors on est oblige de prendre l'objet i
  - si $v_i \geq V$ alors $\overline{OPT}(i, V) = w_i$
  - si $v_i < V$ alors $\overline{OPT}(i, V) = w_i + \overline{OPT}(i-1, V - v_i)$

  Dans tous les cas :
  $\overline{OPT}(i, V) = w_i + \overline{OPT}(i-1, \max\{0, V - v_i\})$
- si $V \leq \sum_{j=1}^{i-1} v_j$
  - soit on prend l'objet i et alors
    $\overline{OPT}(i, V) = w_i + \overline{OPT}(i-1, \max\{0, V - v_i\})$
  - soit on ne le prend pas et alors $\overline{OPT}(i, V) = \overline{OPT}(i-1, V)$

La formule de recurrence

Principe : discussion sur prend on l'objet i dans $\overline{OPT}(i, V)$?

- si $V > \sum_{j=1}^{i-1} v_j$ alors on est oblige de prendre l'objet i
  - si $v_i \geq V$ alors $\overline{OPT}(i, V) = w_i$
  - si $v_i < V$ alors $\overline{OPT}(i, V) = w_i + \overline{OPT}(i-1, V - v_i)$

  Dans tous les cas :
  $\overline{OPT}(i, V) = w_i + \overline{OPT}(i-1, \max\{0, V - v_i\})$
- si $V \leq \sum_{j=1}^{i-1} v_j$
  - soit on prend l'objet i et alors
    $\overline{OPT}(i, V) = w_i + \overline{OPT}(i-1, \max\{0, V - v_i\})$
  - soit on ne le prend pas et alors $\overline{OPT}(i, V) = \overline{OPT}(i-1, V)$

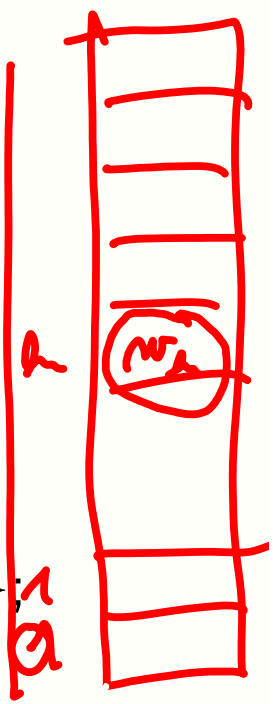
  Dans tous les cas :

$$\overline{OPT}(i, V) = \min\{\overline{OPT}(i-1, V), w_i + \overline{OPT}(i-1, \max\{0, V - v_i\})\}$$

L'algo

Algorithme 1 : Sac a dos a valeurs entieres

1 **pour** i *de* 0 *a* n **faire**
2 $\quad \overline{OPT}(i,0) = 0;$
3 **fin**
4 **pour** i *de* 1 *a* n **faire**
5 $\quad$ **pour** V *de* 1 *a* $\sum_{j=1}^{i} v_j$ **faire**
6 $\quad\quad$ **si** $V > \sum_{j=1}^{i-1} v_j$
7 $\quad\quad$ **alors** $\overline{OPT}(i,V) \leftarrow w_i + \overline{OPT}(i-1, \max\{0, V-v_i\})$;
8 $\quad\quad$ **sinon** $\overline{OPT}(i,V) \leftarrow$
$\quad\quad\quad \min\{\overline{OPT}(i-1,V), w_i + \overline{OPT}(i-1, \max\{0, V-v_i\})\}$;
9 $\quad$ **fin**
10 **fin**
11 **retourner** le V maximum tel que $\overline{OPT}(n,V) \leq W$;



L'algo

Algorithme 1 : Sac a dos a valeurs entieres

1 **pour** i *de* 0 *a* n **faire**
2 | $\overline{OPT}(i,0) = 0$;
3 **fin**
4 **pour** i *de* 1 *a* n **faire**
5 | **pour** V *de* 1 *a* $\sum_{j=1}^{i} v_j$ **faire**
6 | | **si** $V > \sum_{j=1}^{i-1} v_j$
7 | | **alors** $\overline{OPT}(i,V) \leftarrow w_i + \overline{OPT}(i-1, \max\{0, V - v_i\})$;
8 | | **sinon** $\overline{OPT}(i,V) \leftarrow$
| | $\min\{\overline{OPT}(i-1,V), w_i + \overline{OPT}(i-1, \max\{0, V - v_i\})\}$;
9 | **fin**
10 **fin**
11 **retourner** le V maximum tel que $\overline{OPT}(n,V) \leq W$;

L'algo

Algorithme 1 : Sac a dos a valeurs entieres

1 **pour** i *de* 0 *a* n **faire**
2 $\quad \overline{OPT}(i,0) = 0;$
3 **fin**
4 **pour** i *de* 1 *a* n **faire**
5 $\quad$ **pour** V *de* 1 *a* $\sum_{j=1}^{i} v_j$ **faire**
6 $\quad\quad$ **si** $V > \sum_{j=1}^{i-1} v_j$
7 $\quad\quad$ **alors** $\overline{OPT}(i,V) \leftarrow w_i + \overline{OPT}(i-1, \max\{0, V-v_i\})$;
8 $\quad\quad$ **sinon** $\overline{OPT}(i,V) \leftarrow$
$\quad\quad\quad \min\{\overline{OPT}(i-1,V), w_i + \overline{OPT}(i-1, \max\{0, V-v_i\})\}$;
9 $\quad$ **fin**
10 **fin**
11 **retourner** le V maximum tel que $\overline{OPT}(n,V) \leq W$;

dem(i,V) = i

dem(i,V) = dem(i-1,V)

dem(i,V) = i

Algo pour déterminer un sac réalisant le V optimum

$k \leftarrow$ dern (n, V)

$S \leftarrow \{k\}$

$V \leftarrow V - v_k$

tant que $V > 0$ faire

- $k \leftarrow$ dern $(k-1, V)$
- $S \leftarrow S \cup \{k\}$
- $V \leftarrow V - v_k$

retourner S

$k \leftarrow$ dern $(p(k))$

Exemple

v_i	4	1	3	5	2
w_i	2	3	5	4	1

$\overline{OPT}(4,3)$?

$\overline{OPT}(4,8)$?

$\overline{OPT}(4,10)$?

	1	2	3	4	5
8				9	8
7					7
6			10	6	
5			7		
4			7		
3		5	5	4	
2	2	2	2	2	
1	2	2	2	2	
0	2	2	2	2	
	0	0	0	0	0

$W = 7$

$V = 9$

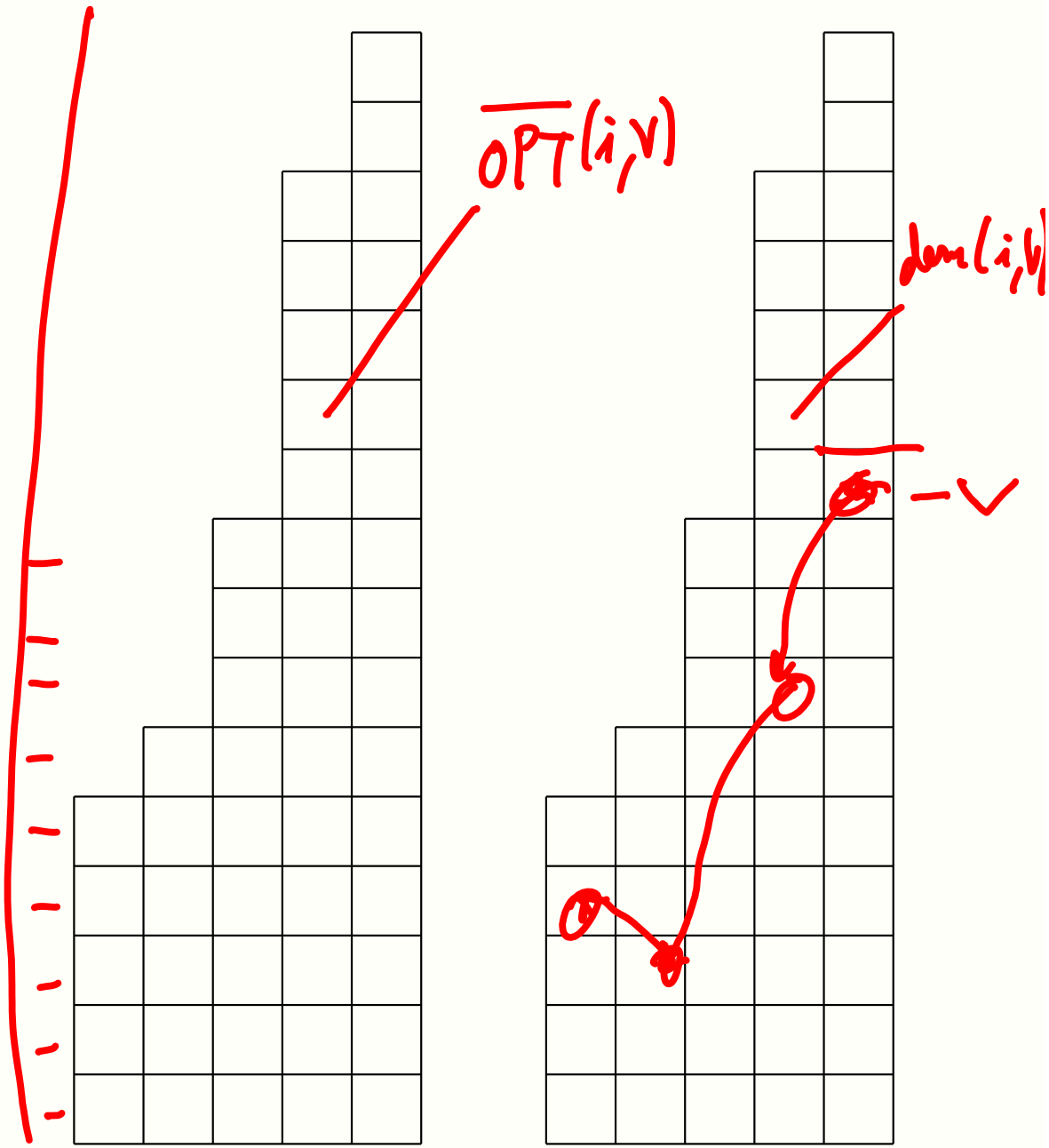
Exemple

v_i	4	1	3	5	2
w_i	2	3	5	4	1

$\overline{OPT}(4,3)$?

$\overline{OPT}(4,8)$?

$\overline{OPT}(4,10)$?



Analyse de complexite

- Boucle ligne 4 : n fois
- Boucle ligne 5 : au plus $\sum_{j=1}^{n} v_j \leq nv^*$ fois
- Complexite totale : $O(n^2 v^*)$

$v^* = \max_p (\log_p v^*)$

Exponentiel en la taille de l'entrée

plus grande valeur codée.

n entiers

n réels

entier v^* : place $(\log(v^*))$

entier p $\rightarrow$ $\log(p)$

111111 ... 111 — pseudo-polynomial.

$n^2 v^*$ → exp (taille de la donnée)

n actions
n réels

v^* : le v max parmi les objets

↓

taille de codage ? pas v^*

$\log v^*$

entier p → 10110001 en binaire # bits : $\log p$

$\log_2(p)$

→ 111 .. 111

p

Analyse de complexite

- Boucle ligne 4 : n fois
- Boucle ligne 5 : au plus $\sum_{j=1}^{n} v_j \leq nv^*$ fois
- Complexite totale : $O(n^2 v^*)$

Attention : cette complexite n'est pas polynomiale mais exponentielle ! ! !

La taille de la donnee est $O(n \log v^*)$ et $v^* = exp(\log v^*)$.

Analyse de complexite

On va utiliser cet algo pour faire un algorithme d'approximation pour le probleme general a valeurs reelles.

Analyse de complexite

On va utiliser cet algo pour faire un algorithme d'approximation pour le probleme general a valeurs reelles.

Pour cela on va modifier les valeurs donnees en entree de sorte que :

- les valeurs $\hat{v}_i$ modifiees soient entieres

Analyse de complexite

On va utiliser cet algo pour faire un algorithme d'approximation pour le probleme general a valeurs reelles.

Pour cela on va modifier les valeurs donnees en entree de sorte que :

- les valeurs $\hat{v}_i$ modifiees soient entieres
- $\hat{v}^* = O(n)$: le temps de calcul sera polynomial

Analyse de complexite

On va utiliser cet algo pour faire un algorithme d'approximation pour le probleme general a valeurs reelles.

Pour cela on va modifier les valeurs donnees en entree de sorte que :

- les valeurs $\hat{v}_i$ modifiees soient entieres
- $\hat{v}^* = O(n)$: le temps de calcul sera polynomial
- optimum du probleme modifie soit une approximation a un facteur $1 + \epsilon$ de l'optimum du probleme initial

L'algo d'approx

- on retire les objets tel que $w_i > W$, qui ne servent a rien

L'algo d'approx

- on retire les objets tel que $w_i > W$, qui ne servent a rien
- on choisit un facteur d'echelle b (on verra comment)

L'algo d'approx

- on retire les objets tel que $w_i > W$, qui ne servent a rien
- on choisit un facteur d'echelle b (on verra comment)
- on arrondit toutes les valeurs v_i sur le multiple de b juste au dessus :

$$\tilde{v}_i = \lceil v_i / b \rceil \cdot b$$

L'algo d'approx

- on retire les objets tel que $w_i > W$, qui ne servent a rien
- on choisit un facteur d'echelle b (on verra comment)
- on arrondit toutes les valeurs v_i sur le multiple de b juste au dessus :

$$\tilde{v}_i = \lceil v_i/b \rceil \cdot b$$

- puis on se ramene a des valeurs entieres en divisant par b :

$$\hat{v}_i = \tilde{v}_i/b = \lceil v_i/b \rceil$$

L'algo d'approx

- on retire les objets tel que $w_i > W$, qui ne servent a rien
- on choisit un facteur d'echelle b (on verra comment)
- on arrondit toutes les valeurs v_i sur le multiple de b juste au dessus :

$$\tilde{v}_i = \lceil v_i / b \rceil \cdot b$$

- puis on se ramene a des valeurs entieres en divisant par b :

$$\hat{v}_i = \tilde{v}_i / b = \lceil v_i / b \rceil$$

- on resoud le probleme sur les $(\hat{v}_i, w_i)$ avec l'algo precedent

L'algo d'approx

- on retire les objets tel que $w_i > W$, qui ne servent a rien
- on choisit un facteur d'echelle b (on verra comment)
- on arrondit toutes les valeurs v_i sur le multiple de b juste au dessus :

$$\tilde{v}_i = \lceil v_i/b \rceil \cdot b$$

- puis on se ramene a des valeurs entieres en divisant par b :

$$\hat{v}_i = \tilde{v}_i/b = \lceil v_i/b \rceil$$

- on resoud le probleme sur les $(\hat{v}_i, w_i)$ avec l'algo precedent

Choix du facteur d'echelle : $b = \frac{\epsilon}{2n} v^*$ (avec $\frac{1}{\epsilon}$ entier)

L'algo d'approx

- on retire les objets tel que $w_i > W$, qui ne servent a rien
- on choisit un facteur d'echelle b (on verra comment)
- on arrondit toutes les valeurs v_i sur le multiple de b juste au dessus :

$$\tilde{v}_i = \lceil v_i/b \rceil \cdot b$$

- puis on se ramene a des valeurs entieres en divisant par b :

$$\hat{v}_i = \tilde{v}_i/b = \lceil v_i/b \rceil$$

- on resoud le probleme sur les $(\hat{v}_i, w_i)$ avec l'algo precedent

Choix du facteur d'echelle : $b = \frac{\epsilon}{2n} v^*$ (avec $\frac{1}{\epsilon}$ entier)

Doit garantir :

- $\hat{v}^* = O(n)$
- un ratio d'approximation de $1 + \epsilon$ pour le probleme initial

Analyse de complexite

$1+\epsilon$

ALGO :

- On fixe ϵ comme on veut.
- On modifie les valeurs de l'instance : $(w_i, \hat{v}_i)$.
- On applique l'algo exact pour le sac a dos a valeurs entieres sur l'instance modifiee : sac a dos optimum S.
- Retourner S

Analyse de complexite

ALGO :

- On fixe ϵ comme on veut.
- On modifie les valeurs de l'instance : $(w_i, \hat{v}_i)$.
- On applique l'algo exact pour le sac a dos a valeurs entieres sur l'instance modifiee : sac a dos optimum S.
- Retourner S

Complexite :

- algo de complexite $O(n^2 v^*)$ applique a l'instance modifiee avec les $(w_i, \hat{v}_i)$: complexite $O(n^2 \hat{v}^*)$

Analyse de complexite

ALGO :

- On fixe ϵ comme on veut.
- On modifie les valeurs de l'instance : $(w_i, \hat{v}_i)$.
- On applique l'algo exact pour le sac a dos a valeurs entieres sur l'instance modifiee : sac a dos optimum S.
- Retourner S

Complexite :

- algo de complexite $O(n^2 v^*)$ applique a l'instance modifiee avec les $(w_i, \hat{v}_i)$: complexite $O(n^2 \hat{v}^*)$
- soit $j \in [\![1, n]\!]$ tel que $v_j = v^*$, on a

$$\hat{v}^* = \hat{v}_j = \lceil v_j / b \rceil = \lceil v^* / b \rceil = \frac{2n}{\epsilon}$$

$O(n)$

Analyse de complexite

ALGO :

- On fixe ϵ comme on veut.
- On modifie les valeurs de l'instance : $(w_i, \hat{v}_i)$.
- On applique l'algo exact pour le sac a dos a valeurs entieres sur l'instance modifiee : sac a dos optimum S.
- Retourner S

Complexite :

- algo de complexite $O(n^2 v^*)$ applique a l'instance modifiee avec les $(w_i, \hat{v}_i)$: complexite $O(n^2 \hat{v}^*)$
- soit $j \in [\![1, n]\!]$ tel que $v_j = v^*$, on a

$$\hat{v}^* = \hat{v}_j = \lceil v_j / b \rceil = \lceil v^* / b \rceil = \frac{2n}{\epsilon}$$

- complexite totale : $O(\frac{1}{\epsilon} n^3) = O(n^3)$ lorsque ϵ est fixe

Le ratio d'approx

- soit S^* la solution optimale du probleme initial et soit S la solution retournee par l'algo d'approx

Le ratio d'approx

- soit S^* la solution optimale du probleme initial et soit S la solution retournee par l'algo d'approx
- comme l'algo resoud le pb avec les $\hat{v}_i$, il le resoud aussi avec les $\tilde{v}_i$ (equivalent a multiplication par b pres)

Le ratio d'approx

- soit S^* la solution optimale du probleme initial et soit S la solution retournee par l'algo d'approx
- comme l'algo resoud le pb avec les $\hat{v}_i$, il le resoud aussi avec les $\tilde{v}_i$ (equivalent a multiplication par b pres)
- on a donc $\sum_{i \in S} \tilde{v}_i \geq \sum_{i \in S^*} \tilde{v}_i$

  (clef : la solution optimale ne peut pas etre trop bonne)

Le ratio d'approx

- soit S^* la solution optimale du probleme initial et soit S la solution retournee par l'algo d'approx
- comme l'algo resoud le pb avec les $\hat{v}_i$, il le resoud aussi avec les $\tilde{v}_i$ (equivalent a multiplication par b pres)
- on a donc $\sum_{i \in S} \tilde{v}_i \geq \sum_{i \in S^*} \tilde{v}_i$
  (clef : la solution optimale ne peut pas etre trop bonne)
- par definition $v_i \leq \tilde{v}_i \leq v_i + b$, d'ou

$$\sum_{i \in S^*} v_i \leq \sum_{i \in S^*} \tilde{v}_i \leq \sum_{i \in S} \tilde{v}_i \leq \sum_{i \in S} (v_i + b) \leq nb + \sum_{i \in S} v_i$$

$\leq (1+\varepsilon) \sum_{i \in S} v_i$ $\qquad$ $\leq \varepsilon \sum_{i \in S} v_i$

Le ratio d'approx

- soit S^* la solution optimale du probleme initial et soit S la solution retournee par l'algo d'approx
- comme l'algo resoud le pb avec les $\hat{v}_i$, il le resoud aussi avec les $\tilde{v}_i$ (equivalent a multiplication par b pres)
- on a donc $\sum_{i \in S} \tilde{v}_i \geq \sum_{i \in S^*} \tilde{v}_i$
  (clef : la solution optimale ne peut pas etre trop bonne)
- par definition $v_i \leq \tilde{v}_i \leq v_i + b$, d'ou

$$\sum_{i \in S^*} v_i \leq \sum_{i \in S^*} \tilde{v}_i \leq \sum_{i \in S} \tilde{v}_i \leq \sum_{i \in S} (v_i + b) \leq nb + \sum_{i \in S} v_i$$

- donc $\sum_{i \in S^*} v_i$ et $\sum_{i \in S} v_i$ different d'au plus nb.

Le ratio d'approx

Montrons que nb est petit devant $\sum_{i \in S} v_i$.

Le ratio d'approx

Montrons que nb est petit devant $\sum_{i \in S} v_i$.

- on reprend la definition de j tel que $v_j = v^*$, on a alors $v_j = \frac{2}{\epsilon} nb$ et $v_j = \tilde{v}_j$.

$$\tilde{v}_i = \left\lceil \frac{v_i}{b} \right\rceil = \left\lceil \frac{2}{\epsilon} n \right\rceil = \frac{2n}{\epsilon}$$

Le ratio d'approx

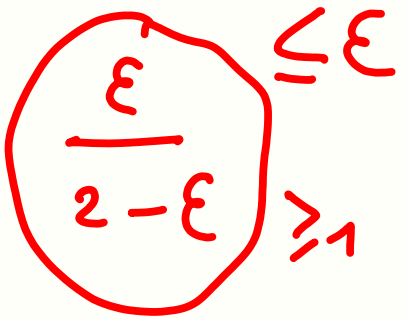
Montrons que nb est petit devant $\sum_{i \in S} v_i$.

- on reprend la definition de j tel que $v_j = v^*$, on a alors $v_j = \frac{2}{\epsilon} nb$ et $v_j = \tilde{v}_j$.
- comme $\forall i, w_i \leq W$, on a $\sum_{i \in S} \tilde{v}_i \geq \tilde{v}_j = \frac{2}{\epsilon} nb$

Le ratio d'approx

Montrons que nb est petit devant $\sum_{i \in S} v_i$.

- on reprend la definition de j tel que $v_j = v^*$, on a alors $v_j = \frac{2}{\epsilon} nb$ et $v_j = \tilde{v}_j$.
- comme $\forall i, w_i \leq W$, on a $\sum_{i \in S} \tilde{v}_i \geq \tilde{v}_j = \frac{2}{\epsilon} nb$
- d'apres les inegalite precedentes on a $\sum_{i \in S} v_i \geq \sum_{i \in S} \tilde{v}_i - nb$, d'ou $\sum_{i \in S} v_i \geq (\frac{2}{\epsilon} - 1) nb$, c.a.d. $nb \leq \frac{1}{\frac{2}{\epsilon} - 1} \cdot \sum_{i \in S} v_i$
- pour $\epsilon \leq 1$, cela donne $nb \leq \epsilon \sum_{i \in S} v_i$



Le ratio d'approx

Montrons que nb est petit devant $\sum_{i \in S} v_i$.

- on reprend la definition de j tel que $v_j = v^*$, on a alors $v_j = \frac{2}{\epsilon} nb$ et $v_j = \tilde{v}_j$.
- comme $\forall i, w_i \leq W$, on a $\sum_{i \in S} \tilde{v}_i \geq \tilde{v}_j = \frac{2}{\epsilon} nb$
- d'apres les inegalite precedentes on a $\sum_{i \in S} v_i \geq \sum_{i \in S} \tilde{v}_i - nb$, d'ou $\sum_{i \in S} v_i \geq (\frac{2}{\epsilon} - 1) nb$, c.a.d. $nb \leq \frac{1}{\frac{2}{\epsilon} - 1} \cdot \sum_{i \in S} v_i$
- pour $\epsilon \leq 1$, cela donne $nb \leq \epsilon \sum_{i \in S} v_i$
- au final, $\sum_{i \in S^*} v_i \leq \sum_{i \in S} v_i + nb \leq (1 + \epsilon) \sum_{i \in S} v_i$

Le ratio d'approx

Montrons que nb est petit devant $\sum_{i \in S} v_i$.

- on reprend la definition de j tel que $v_j = v^*$, on a alors $v_j = \frac{2}{\epsilon} nb$ et $v_j = \tilde{v}_j$.
- comme $\forall i, w_i \leq W$, on a $\sum_{i \in S} \tilde{v}_i \geq \tilde{v}_j = \frac{2}{\epsilon} nb$
- d'apres les inegalite precedentes on a $\sum_{i \in S} v_i \geq \sum_{i \in S} \tilde{v}_i - nb$, d'ou $\sum_{i \in S} v_i \geq (\frac{2}{\epsilon} - 1) nb$, c.a.d. $nb \leq \frac{1}{\frac{2}{\epsilon} - 1} \cdot \sum_{i \in S} v_i$
- pour $\epsilon \leq 1$, cela donne $nb \leq \epsilon \sum_{i \in S} v_i$
- au final, $\sum_{i \in S^*} v_i \leq \sum_{i \in S} v_i + nb \leq (1 + \epsilon) \sum_{i \in S} v_i$

OPT $\leq (1+\epsilon)$ ALG

Conclusion : le ratio d'approx de l'algorithme est donc bien $1 + \epsilon$

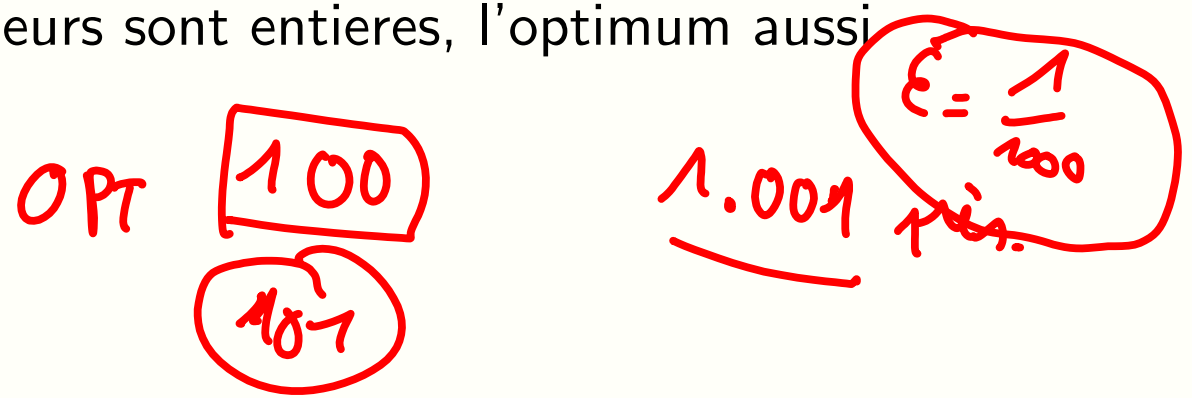
Une idee geniale ! ! !

Un algo polynomial **exact** pour sac a dos a valeurs entieres :
(comme le probleme est NP-complet $\Longrightarrow$ riche et celebre !)

Une idee geniale ! ! !

Un algo polynomial **exact** pour sac a dos a valeurs entieres :
(comme le probleme est NP-complet $\Longrightarrow$ riche et celebre !)

- comme les valeurs sont entieres, l'optimum aussi



Une idee geniale ! ! !

Un algo polynomial **exact** pour sac a dos a valeurs entieres :
(comme le probleme est NP-complet $\Longrightarrow$ riche et celebre !)

- comme les valeurs sont entieres, l'optimum aussi
- comme on approxime a $1+\epsilon$ pres, en prenant ϵ assez petit on obtient la solution exacte, en temps polynomial !

Une idee geniale ! ! !

Un algo polynomial **exact** pour sac a dos a valeurs entieres :
(comme le probleme est NP-complet $\Longrightarrow$ riche et celebre !)

- comme les valeurs sont entieres, l'optimum aussi
- comme on approxime a $1 + \epsilon$ pres, en prenant ϵ assez petit on obtient la solution exacte, en temps polynomial !

Malheureusement :

- il faudrait fixer ϵ assez petit pas seulement pour une instance, mais pour toutes

Une idee geniale ! ! !

Un algo polynomial **exact** pour sac a dos a valeurs entieres :
(comme le probleme est NP-complet $\Longrightarrow$ riche et celebre !)

- comme les valeurs sont entieres, l'optimum aussi
- comme on approxime a $1+\epsilon$ pres, en prenant ϵ assez petit on obtient la solution exacte, en temps polynomial !

Malheureusement :

- il faudrait fixer ϵ assez petit pas seulement pour une instance, mais pour toutes
- et au plus la taille de l'instance est grande, au plus il faut prendre ϵ petit

Une idee geniale ! ! !

Un algo polynomial **exact** pour sac a dos a valeurs entieres :
(comme le probleme est NP-complet $\Longrightarrow$ riche et celebre !)

- comme les valeurs sont entieres, l'optimum aussi
- comme on approxime a $1+\epsilon$ pres, en prenant ϵ assez petit on obtient la solution exacte, en temps polynomial !

Malheureusement :

- il faudrait fixer ϵ assez petit pas seulement pour une instance, mais pour toutes
- et au plus la taille de l'instance est grande, au plus il faut prendre ϵ petit
- en fait, $\frac{1}{\epsilon}$ doit croitre exponentiellement avec la taille

Une idee geniale ! ! !

Un algo polynomial **exact** pour sac a dos a valeurs entieres :
(comme le probleme est NP-complet $\Longrightarrow$ riche et celebre !)

- comme les valeurs sont entieres, l'optimum aussi
- comme on approxime a $1+\epsilon$ pres, en prenant ϵ assez petit on obtient la solution exacte, en temps polynomial !

Malheureusement :

- il faudrait fixer ϵ assez petit pas seulement pour une instance, mais pour toutes
- et au plus la taille de l'instance est grande, au plus il faut prendre ϵ petit
- en fait, $\frac{1}{\epsilon}$ doit croitre exponentiellement avec la taille
- donc la complexite $O(\frac{1}{\epsilon}n^3)$ devient exponentielle...