M1 Info - Optimisation et Recherche Opérationnelle

Cours 5 - Algorithmes d'approximation

Equilibrage de charges et Sac a dos

Semestre Automne 2020-2021 - Université Claude Bernard Lyon 1

Christophe Crespelle
christophe.crespelle@inria.fr

département
Informatique

Faculté des Sciences et Technologies
Université Claude Bernard Lyon 1

Problème de l'equilibrage de charges

Entrée : m machines, n taches avec chacune un temps d'execution de $t_i \in \mathbb{R}_+^*, 1 \leq i \leq n$.

Contraintes :

- taches insecables : effectuees entierement sur une machine
- une machine ne peut faire qu'une tache a la fois

Sortie : une affectation des taches aux m machines telle que la machine qui termine en dernier termine le plus tot possible

Exemple : m=3, n=5, $t_1 = 2$, $t_2 = 3$, $t_3 = 4$, $t_4 = 3$, $t_5 = 5$

Problème de l'equilibrage de charges

Entrée : m machines, n taches avec chacune un temps d'execution de $t_i \in \mathbb{R}_+^*, 1 \leq i \leq n$.

Contraintes :

- taches insecables : effectuees entierement sur une machine
- une machine ne peut faire qu'une tache a la fois

Sortie : une affectation des taches aux m machines telle que la machine qui termine en dernier termine le plus tot possible

Difficulte de calcul : **NP-complet**

Problème de l'equilibrage de charges

Entrée : m machines, n taches avec chacune un temps d'execution de $t_i \in \mathbb{R}_+^*, 1 \leq i \leq n$.

Contraintes :

- taches insecables : effectuees entierement sur une machine
- une machine ne peut faire qu'une tache a la fois

Sortie : une affectation des taches aux m machines telle que la machine qui termine en dernier termine le plus tot possible

Difficulte de calcul : **NP-complet**

On va faire un algorithme polynomial qui donne une solution **approchee garantie**.

Algorithme d'approximation

On note I l'instance du probleme, $OPT(I)$ la valeur de la solution optimale sur I et $ALG(I)$ la valeur retournee par l'algorithme sur I.

Définition

Pour un probleme de minimisation, un algorithme d'approximation avec facteur d'approximation $\rho \geq 1$ est tel que $\forall I, \frac{ALG(I)}{OPT(I)} \leq \rho$.

Pour un probleme de maximisation, un algorithme d'approximation avec facteur d'approximation $\rho \geq 1$ est tel que $\forall I, \frac{OPT(I)}{ALG(I)} \leq \rho$.

Algorithme d'approximation

On note I l'instance du probleme, $OPT(I)$ la valeur de la solution optimale sur I et $ALG(I)$ la valeur retournee par l'algorithme sur I.

Définition

Pour un probleme de minimisation, un algorithme d'approximation avec facteur d'approximation $\rho \geq 1$ est tel que $\forall I, \frac{ALG(I)}{OPT(I)} \leq \rho$.

Pour un probleme de maximisation, un algorithme d'approximation avec facteur d'approximation $\rho \geq 1$ est tel que $\forall I, \frac{OPT(I)}{ALG(I)} \leq \rho$.

Exemple : Un probleme de minimisation et un algorithme pour le resoudre. Sur trois instances I, I', I'', on observe :

- $OPT(I) = 3$ et $ALG(I) = 6$
- $OPT(I') = 8$ et $ALG(I') = 12$
- $OPT(I'') = 6$ et $ALG(I'') = 8$

Quel est le ratio d'approximation de cet algorithme ?

Algorithme d'approximation

On note I l'instance du probleme, $OPT(I)$ la valeur de la solution optimale sur I et $ALG(I)$ la valeur retournee par l'algorithme sur I.

Définition

Pour un probleme de minimisation, un algorithme d'approximation avec facteur d'approximation $\rho \geq 1$ est tel que $\forall I, \frac{ALG(I)}{OPT(I)} \leq \rho$.

Pour un probleme de maximisation, un algorithme d'approximation avec facteur d'approximation $\rho \geq 1$ est tel que $\forall I, \frac{OPT(I)}{ALG(I)} \leq \rho$.

Exemple : pour l'equilibrage de charges on va faire un algo polynomial tel que la solution retournee par l'algorithme verifie toujours $ALG(I) \leq 2OPT(I)$: l'algorithme a un ratio d'approximation 2.

Equilibrage de charges

Définition

Pour $1 \leq i \leq m$, on note $J(i)$ l'ensemble des taches affectees a la machine i.
La charge de la machine i est definie comme $L_i = \sum_{j \in J(i)} t_j$.

Définition

Le **makespan** *d'une affectation est la charge maximum d'une machine dans cette affectation : objectif a minimiser.*
Le makespan minimum sur toutes les affectations sera note L^*.

Premier algorithme - List Scheduling

ALGO :

- considerer les taches une par une dans un **ordre quelconque**
- pour chaque tache l'affecter a la machine la moins chargee

Premier algorithme - List Scheduling

ALGO :

- considerer les taches une par une dans un **ordre quelconque**
- pour chaque tache l'affecter a la machine la moins chargee

Ca c'est simple ! ... et ca donne un ratio d'approximation 2 ! ! !

Premier algorithme - List Scheduling

Conseil general pour prouver les ratio d'algo d'approx :

Montrer que la solution optimale ne peut pas etre trop bonne, pas meilleure que...

Donc dans ce cas il faut minorer L^*.

Premier algorithme - List Scheduling

Conseil general pour prouver les ratio d'algo d'approx :

Montrer que la solution optimale ne peut pas etre trop bonne, pas meilleure que...
Donc dans ce cas il faut minorer L^*.

Lemme

Le makespan minimum $L^* \geq \max_{i=1}^{n}\{t_i\}$.

Premier algorithme - List Scheduling

Conseil general pour prouver les ratio d'algo d'approx :

Montrer que la solution optimale ne peut pas etre trop bonne, pas meilleure que...
Donc dans ce cas il faut minorer L^*.

Lemme

Le makespan minimum $L^* \geq \max_{i=1}^{n}\{t_i\}$.

Lemme

Le makespan minimum $L^* \geq \frac{\sum_{i=1}^{n} t_i}{m}$.

Preuve du ratio d'approx

Théorème

L'algo List Scheduling a un ratio d'approximation 2.

Preuve du ratio d'approx

Théorème

L'algo List Scheduling a un ratio d'approximation 2.

Démonstration.

Soit i machine avec la plus grande charge et soit j la derniere tache qui lui a ete affectee.
On a $\forall k \in [\![1, m]\!], L_i - t_j \leq L_k$.

Preuve du ratio d'approx

Théorème

L'algo List Scheduling a un ratio d'approximation 2.

Démonstration.

Soit i machine avec la plus grande charge et soit j la derniere tache qui lui a ete affectee.

On a $\forall k \in [\![1, m]\!], L_i - t_j \leq L_k$.

D'ou, $(L_i - t_j) \leq \frac{1}{m} \sum_{i=1}^{m} L_i = \frac{1}{m} \sum_{i=1}^{n} t_i$.

□

Preuve du ratio d'approx

Théorème

L'algo List Scheduling a un ratio d'approximation 2.

Démonstration.

Soit i machine avec la plus grande charge et soit j la derniere tache qui lui a ete affectee.

On a $\forall k \in [\![1, m]\!], L_i - t_j \leq L_k$.

D'ou, $(L_i - t_j) \leq \frac{1}{m} \sum_{i=1}^{m} L_i = \frac{1}{m} \sum_{i=1}^{n} t_i$.

C.a.d. $L_i - t_j \leq \frac{1}{m} \sum_{i=1}^{n} t_i \leq L^*$ d'apres le lemme.

□

Preuve du ratio d'approx

Théorème

L'algo List Scheduling a un ratio d'approximation 2.

Démonstration.

Soit i machine avec la plus grande charge et soit j la derniere tache qui lui a ete affectee.

On a $\forall k \in [\![1, m]\!], L_i - t_j \leq L_k$.

D'ou, $(L_i - t_j) \leq \frac{1}{m} \sum_{i=1}^{m} L_i = \frac{1}{m} \sum_{i=1}^{n} t_i$.

C.a.d. $L_i - t_j \leq \frac{1}{m} \sum_{i=1}^{n} t_i \leq L^*$ d'apres le lemme.

Donc, $L_i \leq L^* + t_j \leq 2L^*$, d'apres l'autre lemme. □

A-t-on prouve le meilleur ratio ?

- on a prouve $\rho \leq 2$, mais a-t-on $\rho < 2$?

A-t-on prouve le meilleur ratio ?

- on a prouve $\rho \leq 2$, mais a-t-on $\rho < 2$?
- il faut trouver un exemple ou l'algo donne 2 ou bien s'approche aussi pres qu'on veut de 2

A-t-on prouve le meilleur ratio ?

- on a prouve $\rho \leq 2$, mais a-t-on $\rho < 2$?
- il faut trouver un exemple ou l'algo donne 2 ou bien s'approche aussi pres qu'on veut de 2

Exemple : m machines, $m(m-1)$ taches de duree 1 et une tache de duree m, qui vient en dernier dans l'ordre considere.

A-t-on prouve le meilleur ratio ?

- on a prouve $\rho \leq 2$, mais a-t-on $\rho < 2$?
- il faut trouver un exemple ou l'algo donne 2 ou bien s'approche aussi pres qu'on veut de 2

Exemple : m machines, $m(m-1)$ taches de duree 1 et une tache de duree m, qui vient en dernier dans l'ordre considere.
Par l'algo on obtient $2m-1$ alors qu'on peut faire m.
$2m-1/m \to 2$ quand $m \to +\infty$

A-t-on prouve le meilleur ratio ?

- on a prouve $\rho \leq 2$, mais a-t-on $\rho < 2$?
- il faut trouver un exemple ou l'algo donne 2 ou bien s'approche aussi pres qu'on veut de 2

Exemple : m machines, $m(m-1)$ taches de duree 1 et une tache de duree m, qui vient en dernier dans l'ordre considere.
Par l'algo on obtient $2m-1$ alors qu'on peut faire m.
$2m-1/m \to 2$ quand $m \to +\infty$

- conclusion : le ratio d'approx de cet algo est exactement 2

A-t-on prouve le meilleur ratio ?

- on a prouve $\rho \leq 2$, mais a-t-on $\rho < 2$?
- il faut trouver un exemple ou l'algo donne 2 ou bien s'approche aussi pres qu'on veut de 2

Exemple : m machines, $m(m-1)$ taches de duree 1 et une tache de duree m, qui vient en dernier dans l'ordre considere.
Par l'algo on obtient $2m-1$ alors qu'on peut faire m.
$2m-1/m \to 2$ quand $m \to +\infty$

- conclusion : le ratio d'approx de cet algo est exactement 2

- meilleur ratio avec un autre algo ?

Deuxieme algo : Longest Processing

ALGO :

Meme algo que *List Scheduling* mais avec un ordre d'arrivee des taches bien choisi : par ordre decroissant des durees.

Deuxieme algo : Longest Processing

ALGO :

Meme algo que *List Scheduling* mais avec un ordre d'arrivee des taches bien choisi : par ordre decroissant des durees.

Durees : 2,3,4,3,5. Ordre decroissant : L=(5,4,3,3,2)

Deuxieme algo : Longest Processing

ALGO :

Meme algo que *List Scheduling* mais avec un ordre d'arrivee des taches bien choisi : par ordre decroissant des durees.

Durees : 2,3,4,3,5. Ordre decroissant : L=(5,4,3,3,2)

Analyse :

- on peut supposer que $n > m$, sinon l'algo trouve toujours l'optimal L^*

Deuxieme algo : Longest Processing

ALGO :

Meme algo que *List Scheduling* mais avec un ordre d'arrivee des taches bien choisi : par ordre decroissant des durees.

Durees : 2,3,4,3,5. Ordre decroissant : L=(5,4,3,3,2)

Analyse :

- on peut supposer que $n > m$, sinon l'algo trouve toujours l'optimal L^*
- lorsque t_{m+1} arrive, chaque machine a une charge d'au moins t_m, par consequent $L^* \geq t_m + t_{m+1} \geq 2t_{m+1}$.

Deuxieme algo : Longest Processing

- d'apres l'analyse de l'algo precedent, on a $L_i \leq L^* + t_j$, on peut alors distinguer 2 cas :

Deuxieme algo : Longest Processing

- d'apres l'analyse de l'algo precedent, on a $L_i \leq L^* + t_j$, on peut alors distinguer 2 cas :
  - $j \leq m$: alors L_i n'a qu'une tache, c'est necessairement t_1 et $L^* = t_1$ est trouve par l'algo

Deuxieme algo : Longest Processing

- d'apres l'analyse de l'algo precedent, on a $L_i \leq L^* + t_j$, on peut alors distinguer 2 cas :
  - $j \leq m$: alors L_i n'a qu'une tache, c'est necessairement t_1 et $L^* = t_1$ est trouve par l'algo
  - $j \geq m+1$: alors $t_j \leq t_{m+1} \leq L^*/2$ et donc $L_i \leq 3/2 \cdot L^*$.

Deuxieme algo : Longest Processing

- d'apres l'analyse de l'algo precedent, on a $L_i \leq L^* + t_j$, on peut alors distinguer 2 cas :
  - $j \leq m$: alors L_i n'a qu'une tache, c'est necessairement t_1 et $L^* = t_1$ est trouve par l'algo
  - $j \geq m+1$: alors $t_j \leq t_{m+1} \leq L^*/2$ et donc $L_i \leq 3/2 \cdot L^*$.
- conclusion : $\rho \leq 3/2$

Deuxieme algo : Longest Processing

- d'apres l'analyse de l'algo precedent, on a $L_i \leq L^* + t_j$, on peut alors distinguer 2 cas :
  - $j \leq m$: alors L_i n'a qu'une tache, c'est necessairement t_1 et $L^* = t_1$ est trouve par l'algo
  - $j \geq m+1$: alors $t_j \leq t_{m+1} \leq L^*/2$ et donc $L_i \leq 3/2 \cdot L^*$.
- conclusion : $\rho \leq 3/2$
- A-t-on $\rho < 3/2$?

Deuxieme algo : Longest Processing

- d'apres l'analyse de l'algo precedent, on a $L_i \leq L^* + t_j$, on peut alors distinguer 2 cas :
  - $j \leq m$: alors L_i n'a qu'une tache, c'est necessairement t_1 et $L^* = t_1$ est trouve par l'algo
  - $j \geq m+1$: alors $t_j \leq t_{m+1} \leq L^*/2$ et donc $L_i \leq 3/2 \cdot L^*$.
- conclusion : $\rho \leq 3/2$
- A-t-on $\rho < 3/2$?
- Oui. On peut montrer $\rho \leq 4/3$.

Deuxieme algo : Longest Processing

- d'apres l'analyse de l'algo precedent, on a $L_i \leq L^* + t_j$, on peut alors distinguer 2 cas :
  - $j \leq m$: alors L_i n'a qu'une tache, c'est necessairement t_1 et $L^* = t_1$ est trouve par l'algo
  - $j \geq m+1$: alors $t_j \leq t_{m+1} \leq L^*/2$ et donc $L_i \leq 3/2 \cdot L^*$.
- conclusion : $\rho \leq 3/2$
- A-t-on $\rho < 3/2$?
- Oui. On peut montrer $\rho \leq 4/3$.
  Et c'est le mieux qu'on puisse montrer :
  m machines, $n = 2m+1$ taches, 2 taches de duree $m+1$, $m+2$, ..., $2m$ et une de duree m.

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

Difficulte de calcul : **NP-complet**

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

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 $\overline{OPT}(n,V) \leq W;$

Analyse de complexite

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

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

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

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

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

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

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 $v_j = \tilde{v}_j$.

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, $\sum_{i \in S^*} v_i \leq \sum_{i \in S} v_i + nb \leq (1 + \epsilon) \sum_{i \in S} v_i$

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...