

Cours 5: Algorithmes approchés

Algorithmes approchés

Schémas d'approximation

Bin-packing: limite de l'approximabilité.

Voyageur de commerce : non approximabilité

Cours 5: Algorithmes approchés

Algorithmes approchés

Schémas d'approximation

Bin-packing: limite de l'approximabilité.

Voyageur de commerce : non approximabilité

Un algorithme glouton pour COUVRANT

Donnée: Un graphe $G = (X, E)$ non-orienté

Problème: Trouver un ensemble minimum de sommets couvrant les arêtes.

Un algorithme glouton pour COUVRANT

Donnée: Un graphe $G = (X, E)$ non-orienté NP-complet !

Problème: Trouver un ensemble minimum de sommets couvrant les arêtes.

Un algorithme glouton pour COUVRANT

Donnée: Un graphe $G = (X, E)$ non-orienté **NP-complet !**

Problème: Trouver un ensemble minimum de sommets couvrant les arêtes.

Algorithme: $F := E; Y := \emptyset$

Tant que F est non vide: Choisir une arête f dans F

 ajouter ses 2 extrémités dans Y

 et supprimer de F toutes les arêtes couvertes par ces 2 sommets.

Un algorithme glouton pour COUVRANT

Donnée: Un graphe $G = (X, E)$ non-orienté **NP-complet !**

Problème: Trouver un ensemble minimum de sommets couvrant les arêtes.

Algorithme: $F := E; Y := \emptyset$

Tant que F est non vide: Choisir une arête f dans F
ajouter ses 2 extrémités dans Y
et supprimer de F toutes les arêtes couvertes par ces 2 sommets.

Non optimalité: pour un graphe réduit à une arête, 2 sommets au lieu d'1

Un algorithme glouton pour COUVRANT

Donnée: Un graphe $G = (X, E)$ non-orienté **NP-complet !**

Problème: Trouver un ensemble minimum de sommets couvrant les arêtes.

Algorithme: $F := E; Y := \emptyset$

Tant que F est non vide: Choisir une arête f dans F
ajouter ses 2 extrémités dans Y
et supprimer de F toutes les arêtes couvertes par ces 2 sommets.

Non optimalité: pour un graphe réduit à une arête, 2 sommets au lieu d'1

L'algo peut il être encore plus mauvais que ce facteur 2 ?

Un algorithme glouton pour COUVRANT

Donnée: Un graphe $G = (X, E)$ non-orienté **NP-complet !**

Problème: Trouver un ensemble minimum de sommets couvrant les arêtes.

Algorithme: $F := E; Y := \emptyset$

Tant que F est non vide: Choisir une arête f dans F
ajouter ses 2 extrémités dans Y
et supprimer de F toutes les arêtes couvertes par ces 2 sommets.

Non optimalité: pour un graphe réduit à une arête, 2 sommets au lieu d'1

L'algo peut il être encore plus mauvais que ce facteur 2 ?

Remarque: si H est un couplage et C un couvrant de G alors $|H| < |C|$

Un algorithme glouton pour COUVRANT

Donnée: Un graphe $G = (X, E)$ non-orienté **NP-complet !**

Problème: Trouver un ensemble minimum de sommets couvrant les arêtes.

Algorithme: $F := E; Y := \emptyset$

Tant que F est non vide: Choisir une arête f dans F
ajouter ses 2 extrémités dans Y
et supprimer de F toutes les arêtes couvertes par ces 2 sommets.

Non optimalité: pour un graphe réduit à une arête, 2 sommets au lieu d'1

L'algo peut il être encore plus mauvais que ce facteur 2 ?

Remarque: si H est un couplage et C un couvrant de G alors $|H| < |C|$

Or les arêtes utilisées pour constituer Y forment un couplage:

$$|Y|/2 < |C_{\text{opt}}|$$

Un algorithme glouton pour COUVRANT

Donnée: Un graphe $G = (X, E)$ non-orienté **NP-complet !**

Problème: Trouver un ensemble minimum de sommets couvrant les arêtes.

Algorithme: $F := E; Y := \emptyset$

Tant que F est non vide: Choisir une arête f dans F
ajouter ses 2 extrémités dans Y
et supprimer de F toutes les arêtes couvertes par ces 2 sommets.

Non optimalité: pour un graphe réduit à une arête, 2 sommets au lieu d'1

L'algo peut il être encore plus mauvais que ce facteur 2 ?

Remarque: si H est un couplage et C un couvrant de G alors $|H| < |C|$

Or les arêtes utilisées pour constituer Y forment un couplage:

$$|Y|/2 < |C_{\text{opt}}|$$

\Rightarrow au pire on aura 2 fois trop de sommets

Algorithmes approchés

Problème: Trouver parmi les solutions d'un problème celle qui optimise une fonction f .

Algorithmes approchés

Problème: Trouver parmi les solutions d'un problème celle qui optimise une fonction f .

Un algorithme est ε -approché s'il donne un X qui satisfait

Algorithmes approchés

Problème: Trouver parmi les solutions d'un problème celle qui optimise une fonction f .

Un algorithme est ε -**approché** s'il donne un X qui satisfait

$$\frac{f(X_{\text{opt}}) - f(X)}{f(X_{\text{opt}})} \leq \varepsilon \quad \text{si } X_{\text{opt}} \text{ maximise } f$$

$$\frac{f(X) - f(X_{\text{opt}})}{f(X)} \leq \varepsilon \quad \text{si } X_{\text{opt}} \text{ minimise } f$$

Algorithmes approchés

Problème: Trouver parmi les solutions d'un problème celle qui optimise une fonction f .

Un algorithme est ε -**approché** s'il donne un X qui satisfait

$$\frac{f(X_{\text{opt}}) - f(X)}{f(X_{\text{opt}})} \leq \varepsilon \quad \text{si } X_{\text{opt}} \text{ maximise } f$$

On veut ε proche de 0.

$$\frac{f(X) - f(X_{\text{opt}})}{f(X)} \leq \varepsilon \quad \text{si } X_{\text{opt}} \text{ minimise } f$$

Algorithmes approchés

Problème: Trouver parmi les solutions d'un problème celle qui optimise une fonction f .

Un algorithme est ε -**approché** s'il donne un X qui satisfait

$$\frac{f(X_{\text{opt}}) - f(X)}{f(X_{\text{opt}})} \leq \varepsilon \quad \text{si } X_{\text{opt}} \text{ maximise } f$$

On veut ε proche de 0.

$$\frac{f(X) - f(X_{\text{opt}})}{f(X)} \leq \varepsilon \quad \text{si } X_{\text{opt}} \text{ minimise } f$$

Un algorithme approche l'optimum à un **facteur** θ si

$$\max\left(\frac{f(X)}{f(X_{\text{opt}})}, \frac{f(X_{\text{opt}})}{f(X)}\right) \leq \theta$$

On veut θ proche de 1.

Algorithmes approchés

Problème: Trouver parmi les solutions d'un problème celle qui optimise une fonction f .

Un algorithme est ε -approché s'il donne un X qui satisfait

$$\frac{f(X_{\text{opt}}) - f(X)}{f(X_{\text{opt}})} \leq \varepsilon \quad \text{si } X_{\text{opt}} \text{ maximise } f \quad \text{On veut } \varepsilon \text{ proche de } 0.$$

$$\frac{f(X) - f(X_{\text{opt}})}{f(X)} \leq \varepsilon \quad \text{si } X_{\text{opt}} \text{ minimise } f$$

Un algorithme approche l'optimum à un facteur θ si

$$\max\left(\frac{f(X)}{f(X_{\text{opt}})}, \frac{f(X_{\text{opt}})}{f(X)}\right) \leq \theta \quad \text{On veut } \theta \text{ proche de } 1.$$

Deux formulations équivalentes utilisées indifféremment.

Un algorithme glouton pour COUVRANT

Donnée: Un graphe $G = (X, E)$ non-orienté **NP-complet !**

Problème: Trouver un ensemble minimum de sommets couvrant les arêtes.

Algorithme: $F := E; Y := \emptyset$

Tant que F est non vide: Choisir une arête f dans F
ajouter ses 2 extrémités dans Y
et supprimer de F toutes les arêtes couvertes par ces 2 sommets.

L'ensemble couvrant Y produit vérifie: $|Y| < 2|C_{\text{opt}}|$

D'où $\varepsilon = \frac{|Y| - |C_{\text{opt}}|}{|Y|} \leq \frac{1}{2}$, ou encore $\theta = \frac{|Y|}{|C_{\text{opt}}|} \leq 2$.

\Rightarrow Cet algorithme est $\frac{1}{2}$ -approché, ou approché à un facteur 2.

Un algorithme glouton pour COUVRANT

Donnée: Un graphe $G = (X, E)$ non-orienté **NP-complet !**

Problème: Trouver un ensemble minimum de sommets couvrant les arêtes.

Algorithme: $F := E; Y := \emptyset$

Tant que F est non vide: Choisir une arête f dans F
ajouter ses 2 extrémités dans Y
et supprimer de F toutes les arêtes couvertes par ces 2 sommets.

L'ensemble couvrant Y produit vérifie: $|Y| < 2|C_{\text{opt}}|$

D'où $\varepsilon = \frac{|Y| - |C_{\text{opt}}|}{|Y|} \leq \frac{1}{2}$, ou encore $\theta = \frac{|Y|}{|C_{\text{opt}}|} \leq 2$.

\Rightarrow Cet algorithme est $\frac{1}{2}$ -approché, ou approché à un facteur 2.

La classe des problème d'optimisation pour lesquels il existe un algorithme **polynomial** d'approximation à un **facteur borné** est notée **APX**.

Cours 5: Algorithmes approchés

Algorithmes approchés

Schémas d'approximation

Bin-packing: limite de l'approximabilité.

Voyageur de commerce : non approximabilité

Sac à dos et schéma d'approximation polynomial

Donnée: des poids p_1, \dots, p_n , des gains a_1, \dots, a_p et la capacité P

Problème: Trouver $\max (\sum_i a_i x_i \mid \sum_i p_i x_i \leq P, x_i \in \{0, 1\})$

Sac à dos et schéma d'approximation polynomial

Donnée: des poids p_1, \dots, p_n , des gains a_1, \dots, a_p et la capacité P

Problème: Trouver $\max (\sum_i a_i x_i \mid \sum_i p_i x_i \leq P, x_i \in \{0, 1\})$

Algorithme glouton exact pour x_i réels:

- Réordonner les objets de sorte que $\frac{a_1}{p_1} \geq \frac{a_2}{p_2} \geq \dots \geq \frac{a_n}{p_n}$.
- $s := 0$; Pour $j := 1$ à n faire
 - si $s + p_j \leq P$ faire $s := s + p_j$ et $x_j := 1$
 - sinon $x_j := \frac{P-s}{p_j}$, $x_k := 0$ pour $k > j$ et sortir de la boucle.

Sac à dos et schéma d'approximation polynomial

Donnée: des poids p_1, \dots, p_n , des gains a_1, \dots, a_p et la capacité P

Problème: Trouver $\max (\sum_i a_i x_i \mid \sum_i p_i x_i \leq P, x_i \in \{0, 1\})$

Algorithme glouton exact pour x_i réels:

- Réordonner les objets de sorte que $\frac{a_1}{p_1} \geq \frac{a_2}{p_2} \geq \dots \geq \frac{a_n}{p_n}$.
- $s := 0$; Pour $j := 1$ à n faire
 - si $s + p_j \leq P$ faire $s := s + p_j$ et $x_j := 1$
 - sinon $x_j := \frac{P-s}{p_j}$, $x_k := 0$ pour $k > j$ et sortir de la boucle.

La dernière étape j_* complète le sac: $\sum p_j x_j = P$ (sauf si $P > \sum p_j$)

et l'optimum est $A_* = \sum a_j x_j = \sum_{j < j_*} a_j + \frac{P - \sum_{j < j_*} p_j}{p_{j_*}} a_{j_*}$.

Sac à dos et schéma d'approximation polynomial

Donnée: des poids p_1, \dots, p_n , des gains a_1, \dots, a_p et la capacité P

Problème: Trouver $\max (\sum_i a_i x_i \mid \sum_i p_i x_i \leq P, x_i \in \{0, 1\})$

Algorithme glouton exact pour x_i réels:

- Réordonner les objets de sorte que $\frac{a_1}{p_1} \geq \frac{a_2}{p_2} \geq \dots \geq \frac{a_n}{p_n}$.
- $s := 0$; Pour $j := 1$ à n faire
 - si $s + p_j \leq P$ faire $s := s + p_j$ et $x_j := 1$
 - sinon $x_j := \frac{P-s}{p_j}$, $x_k := 0$ pour $k > j$ et sortir de la boucle.

La dernière étape j_* complète le sac: $\sum p_j x_j = P$ (sauf si $P > \sum p_j$)

et l'optimum est $A_* = \sum a_j x_j = \sum_{j < j_*} a_j + \frac{P - \sum_{j < j_*} p_j}{p_{j_*}} a_{j_*}$.

Algorithme glouton naïf pour x_i entiers: idem sauf la dernière étape où on ne prend rien.

Sac à dos et schéma d'approximation polynomial

Donnée: des poids p_1, \dots, p_n , des gains a_1, \dots, a_p et la capacité P

Problème: Trouver $\max (\sum_i a_i x_i \mid \sum_i p_i x_i \leq P, x_i \in \{0, 1\})$

Algorithme glouton exact pour x_i réels:

- Réordonner les objets de sorte que $\frac{a_1}{p_1} \geq \frac{a_2}{p_2} \geq \dots \geq \frac{a_n}{p_n}$.
- $s := 0$; Pour $j := 1$ à n faire
 - si $s + p_j \leq P$ faire $s := s + p_j$ et $x_j := 1$
 - sinon $x_j := \frac{P-s}{p_j}$, $x_k := 0$ pour $k > j$ et sortir de la boucle.

La dernière étape j_* complète le sac: $\sum p_j x_j = P$ (sauf si $P > \sum p_j$)

et l'optimum est $A_* = \sum a_j x_j = \sum_{j < j_*} a_j + \frac{P - \sum_{j < j_*} p_j}{p_{j_*}} a_{j_*}$.

Algorithme glouton naïf pour x_i entiers: idem sauf la dernière étape où on ne prend rien.

C'est arbitrairement mauvais même avec 2 objets: si $p_2 = (k + 1)p_1$, $a_2 = ka_1$ et $P = p_2$, alors on a bien $\frac{a_1}{p_1} \geq \frac{a_2}{p_2}$ et on perd un facteur k .

Sac à dos et schéma d'approximation polynomial

Donnée: des poids p_1, \dots, p_n , des gains a_1, \dots, a_p et la capacité P

Problème: Trouver $\max \left(\sum_i a_i x_i \mid \sum_i p_i x_i \leq P, x_i \in \{0, 1\} \right)$

Amélioration triviale du glouton: On choisit le meilleur gain entre la solution du glouton et $\max(a_i)$ (on suppose que $p_i \leq P$ pour tout i).

Sac à dos et schéma d'approximation polynomial

Donnée: des poids p_1, \dots, p_n , des gains a_1, \dots, a_p et la capacité P

Problème: Trouver $\max (\sum_i a_i x_i \mid \sum_i p_i x_i \leq P, x_i \in \{0, 1\})$

Amélioration triviale du glouton: On choisit le meilleur gain entre la solution du glouton et $\max(a_i)$ (on suppose que $p_i \leq P$ pour tout i).

Théorème: Le glouton amélioré trouve une solution dont le bénéfice A_+ est au moins la moitié de l'optimum entier A_{opt} .

Sac à dos et schéma d'approximation polynomial

Donnée: des poids p_1, \dots, p_n , des gains a_1, \dots, a_p et la capacité P

Problème: Trouver $\max (\sum_i a_i x_i \mid \sum_i p_i x_i \leq P, x_i \in \{0, 1\})$

Amélioration triviale du glouton: On choisit le meilleur gain entre la solution du glouton et $\max(a_i)$ (on suppose que $p_i \leq P$ pour tout i).

Théorème: Le glouton amélioré trouve une solution dont le bénéfice A_+ est au moins la moitié de l'optimum entier A_{opt} .

Preuve: L'optimal entier A_{opt} est au plus l'optimal réel:

$$A_{\text{opt}} \leq A_* = \sum_{j < j_*} a_j + \frac{P - \sum_{j < j_*} p_j}{p_{j_*}} a_{j_*}$$

Sac à dos et schéma d'approximation polynomial

Donnée: des poids p_1, \dots, p_n , des gains a_1, \dots, a_p et la capacité P

Problème: Trouver $\max (\sum_i a_i x_i \mid \sum_i p_i x_i \leq P, x_i \in \{0, 1\})$

Amélioration triviale du glouton: On choisit le meilleur gain entre la solution du glouton et $\max(a_i)$ (on suppose que $p_i \leq P$ pour tout i).

Théorème: Le glouton amélioré trouve une solution dont le bénéfice A_+ est au moins la moitié de l'optimum entier A_{opt} .

Preuve: L'optimal entier A_{opt} est au plus l'optimal réel:

$$A_{\text{opt}} \leq A_* = \sum_{j < j_*} a_j + \frac{P - \sum_{j < j_*} p_j}{p_{j_*}} a_{j_*}$$

Comme on s'est arrêté sur j_* , on a $P < \sum_{j < j_*} p_j + p_{j_*}$.

Sac à dos et schéma d'approximation polynomial

Donnée: des poids p_1, \dots, p_n , des gains a_1, \dots, a_p et la capacité P

Problème: Trouver $\max (\sum_i a_i x_i \mid \sum_i p_i x_i \leq P, x_i \in \{0, 1\})$

Amélioration triviale du glouton: On choisit le meilleur gain entre la solution du glouton et $\max(a_i)$ (on suppose que $p_i \leq P$ pour tout i).

Théorème: Le glouton amélioré trouve une solution dont le bénéfice A_+ est au moins la moitié de l'optimum entier A_{opt} .

Preuve: L'optimal entier A_{opt} est au plus l'optimal réel:

$$A_{\text{opt}} \leq A_* = \sum_{j < j_*} a_j + \frac{P - \sum_{j < j_*} p_j}{p_{j_*}} a_{j_*} \leq \sum_{j < j_*} a_j + a_{j_*}.$$

Comme on s'est arrêté sur j_* , on a $P < \sum_{j < j_*} p_j + p_{j_*}$.

Sac à dos et schéma d'approximation polynomial

Donnée: des poids p_1, \dots, p_n , des gains a_1, \dots, a_p et la capacité P

Problème: Trouver $\max (\sum_i a_i x_i \mid \sum_i p_i x_i \leq P, x_i \in \{0, 1\})$

Amélioration triviale du glouton: On choisit le meilleur gain entre la solution du glouton et $\max(a_i)$ (on suppose que $p_i \leq P$ pour tout i).

Théorème: Le glouton amélioré trouve une solution dont le bénéfice A_+ est au moins la moitié de l'optimum entier A_{opt} .

Preuve: L'optimal entier A_{opt} est au plus l'optimal réel:

$$A_{\text{opt}} \leq A_* = \sum_{j < j_*} a_j + \frac{P - \sum_{j < j_*} p_j}{p_{j_*}} a_{j_*} \leq \sum_{j < j_*} a_j + a_{j_*}.$$

Comme on s'est arrêté sur j_* , on a $P < \sum_{j < j_*} p_j + p_{j_*}$.

Or le glouton amélioré fait au moins $\sum_{j < j_*} a_j$ (glouton) ou a_{j_*} .

Sac à dos et schéma d'approximation polynomial

Donnée: des poids p_1, \dots, p_n , des gains a_1, \dots, a_p et la capacité P

Problème: Trouver $\max (\sum_i a_i x_i \mid \sum_i p_i x_i \leq P, x_i \in \{0, 1\})$

Amélioration triviale du glouton: On choisit le meilleur gain entre la solution du glouton et $\max(a_i)$ (on suppose que $p_i \leq P$ pour tout i).

Théorème: Le glouton amélioré trouve une solution dont le bénéfice A_+ est au moins la moitié de l'optimum entier A_{opt} .

Preuve: L'optimal entier A_{opt} est au plus l'optimal réel:

$$A_{\text{opt}} \leq A_* = \sum_{j < j_*} a_j + \frac{P - \sum_{j < j_*} p_j}{p_{j_*}} a_{j_*} \leq \sum_{j < j_*} a_j + a_{j_*}.$$

Comme on s'est arrêté sur j_* , on a $P < \sum_{j < j_*} p_j + p_{j_*}$.

Or le glouton amélioré fait au moins $\sum_{j < j_*} a_j$ (glouton) ou a_{j_*} .

Glouton amélioré est $\frac{1}{2}$ -approché !

Sac à dos et schéma d'approximation polynomial

Donnée: des poids p_1, \dots, p_n , des gains a_1, \dots, a_p et la capacité P

Problème: Trouver $\max (\sum_i a_i x_i \mid \sum_i p_i x_i \leq P, x_i \in \{0, 1\})$

Théorème (Schéma d'approximation polynomial): Pour tout ε il existe un algorithme ε -approché pour le problème SAC-À-DOS de complexité $O(n^3/\varepsilon)$.

Sac à dos et schéma d'approximation polynomial

Donnée: des poids p_1, \dots, p_n , des gains a_1, \dots, a_p et la capacité P

Problème: Trouver $\max (\sum_i a_i x_i \mid \sum_i p_i x_i \leq P, x_i \in \{0, 1\})$

Théorème (Schéma d'approximation polynomial): Pour tout ε il existe un algorithme ε -approché pour le problème SAC-À-DOS de complexité $O(n^3/\varepsilon)$.

Idée: • On utilise la programmation dynamique sur des données arrondies:

Sac à dos et schéma d'approximation polynomial

Donnée: des poids p_1, \dots, p_n , des gains a_1, \dots, a_p et la capacité P

Problème: Trouver $\max \left(\sum_i a_i x_i \mid \sum_i p_i x_i \leq P, x_i \in \{0, 1\} \right)$

Théorème (Schéma d'approximation polynomial): Pour tout ε il existe un algorithme ε -approché pour le problème SAC-À-DOS de complexité $O(n^3/\varepsilon)$.

Idée: • On utilise la programmation dynamique sur des données arrondies:

On pose $b_i = \lfloor \frac{a_i}{10^k} \rfloor$ pour un k bien choisi.

Par programmation dynamique, on obtient l'optimum $\{x'_i\}$ du problème arrondi en temps $O(n^2 M') = O(\frac{n^2 M}{10^k})$ où $M = \max(a_i)$.

Sac à dos et schéma d'approximation polynomial

Donnée: des poids p_1, \dots, p_n , des gains a_1, \dots, a_p et la capacité P

Problème: Trouver $\max \left(\sum_i a_i x_i \mid \sum_i p_i x_i \leq P, x_i \in \{0, 1\} \right)$

Théorème (Schéma d'approximation polynomial): Pour tout ε il existe un algorithme ε -approché pour le problème SAC-À-DOS de complexité $O(n^3/\varepsilon)$.

Idée: ● On utilise la programmation dynamique sur des données arrondies:

On pose $b_i = \lfloor \frac{a_i}{10^k} \rfloor$ pour un k bien choisi.

Par programmation dynamique, on obtient l'optimum $\{x'_i\}$ du problème arrondi en temps $O(n^2 M') = O(\frac{n^2 M}{10^k})$ où $M = \max(a_i)$.

● On compare l'approximation du problème exact donnée par $\{x'_i\}$ à un optimum $\{x_i\}$:

Sac à dos et schéma d'approximation polynomial

Donnée: des poids p_1, \dots, p_n , des gains a_1, \dots, a_p et la capacité P

Problème: Trouver $\max \left(\sum_i a_i x_i \mid \sum_i p_i x_i \leq P, x_i \in \{0, 1\} \right)$

Théorème (Schéma d'approximation polynomial): Pour tout ε il existe un algorithme ε -approché pour le problème SAC-À-DOS de complexité $O(n^3/\varepsilon)$.

Idée: • On utilise la programmation dynamique sur des données arrondies:

On pose $b_i = \lfloor \frac{a_i}{10^k} \rfloor$ pour un k bien choisi.

Par programmation dynamique, on obtient l'optimum $\{x'_i\}$ du problème arrondi en temps $O(n^2 M') = O(\frac{n^2 M}{10^k})$ où $M = \max(a_i)$.

• On compare l'approximation du problème exact donnée par $\{x'_i\}$ à un optimum $\{x_i\}$:

$$\sum_i a_i x'_i \geq \sum_i 10^k b_i x'_i$$

Sac à dos et schéma d'approximation polynomial

Donnée: des poids p_1, \dots, p_n , des gains a_1, \dots, a_p et la capacité P

Problème: Trouver $\max \left(\sum_i a_i x_i \mid \sum_i p_i x_i \leq P, x_i \in \{0, 1\} \right)$

Théorème (Schéma d'approximation polynomial): Pour tout ε il existe un algorithme ε -approché pour le problème SAC-À-DOS de complexité $O(n^3/\varepsilon)$.

Idée: • On utilise la programmation dynamique sur des données arrondies:

On pose $b_i = \lfloor \frac{a_i}{10^k} \rfloor$ pour un k bien choisi.

Par programmation dynamique, on obtient l'optimum $\{x'_i\}$ du problème arrondi en temps $O(n^2 M') = O(\frac{n^2 M}{10^k})$ où $M = \max(a_i)$.

• On compare l'approximation du problème exact donnée par $\{x'_i\}$ à un optimum $\{x_i\}$:

$$\sum_i a_i x'_i \geq \sum_i 10^k b_i x'_i \geq \sum_i 10^k b_i x_i$$

Sac à dos et schéma d'approximation polynomial

Donnée: des poids p_1, \dots, p_n , des gains a_1, \dots, a_p et la capacité P

Problème: Trouver $\max (\sum_i a_i x_i \mid \sum_i p_i x_i \leq P, x_i \in \{0, 1\})$

Théorème (Schéma d'approximation polynomial): Pour tout ε il existe un algorithme ε -approché pour le problème SAC-À-DOS de complexité $O(n^3/\varepsilon)$.

Idée: • On utilise la programmation dynamique sur des données arrondies:

On pose $b_i = \lfloor \frac{a_i}{10^k} \rfloor$ pour un k bien choisi.

Par programmation dynamique, on obtient l'optimum $\{x'_i\}$ du problème arrondi en temps $O(n^2 M') = O(\frac{n^2 M}{10^k})$ où $M = \max(a_i)$.

• On compare l'approximation du problème exact donnée par $\{x'_i\}$ à un optimum $\{x_i\}$:

$$\sum_i a_i x'_i \geq \sum_i 10^k b_i x'_i \geq \sum_i 10^k b_i x_i \geq \sum_i (a_i - 10^k) x_i$$

Sac à dos et schéma d'approximation polynomial

Donnée: des poids p_1, \dots, p_n , des gains a_1, \dots, a_p et la capacité P

Problème: Trouver $\max (\sum_i a_i x_i \mid \sum_i p_i x_i \leq P, x_i \in \{0, 1\})$

Théorème (Schéma d'approximation polynomial): Pour tout ε il existe un algorithme ε -approché pour le problème SAC-À-DOS de complexité $O(n^3/\varepsilon)$.

Idée: • On utilise la programmation dynamique sur des données arrondies:

On pose $b_i = \lfloor \frac{a_i}{10^k} \rfloor$ pour un k bien choisi.

Par programmation dynamique, on obtient l'optimum $\{x'_i\}$ du problème arrondi en temps $O(n^2 M') = O(\frac{n^2 M}{10^k})$ où $M = \max(a_i)$.

• On compare l'approximation du problème exact donnée par $\{x'_i\}$ à un optimum $\{x_i\}$:

$$\sum_i a_i x'_i \geq \sum_i 10^k b_i x'_i \geq \sum_i 10^k b_i x_i \geq \sum_i (a_i - 10^k) x_i \geq \sum_i a_i x_i - n 10^k$$

Sac à dos et schéma d'approximation polynomial

Donnée: des poids p_1, \dots, p_n , des gains a_1, \dots, a_p et la capacité P

Problème: Trouver $\max \left(\sum_i a_i x_i \mid \sum_i p_i x_i \leq P, x_i \in \{0, 1\} \right)$

Théorème (Schéma d'approximation polynomial): Pour tout ε il existe un algorithme ε -approché pour le problème SAC-À-DOS de complexité $O(n^3/\varepsilon)$.

Idée: • On utilise la programmation dynamique sur des données arrondies:

On pose $b_i = \lfloor \frac{a_i}{10^k} \rfloor$ pour un k bien choisi.

Par programmation dynamique, on obtient l'optimum $\{x'_i\}$ du problème arrondi en temps $O(n^2 M') = O(\frac{n^2 M}{10^k})$ où $M = \max(a_i)$.

• On compare l'approximation du problème exact donnée par $\{x'_i\}$ à un optimum $\{x_i\}$:

$$\sum_i a_i x'_i \geq \sum_i 10^k b_i x'_i \geq \sum_i 10^k b_i x_i \geq \sum_i (a_i - 10^k) x_i \geq \sum_i a_i x_i - n 10^k$$

Si $k \leq \log_{10} \frac{A_0 \varepsilon}{n} \leq \log_{10} \frac{M \varepsilon}{n}$, on a une ε -approximation; temps $O(\frac{n^3}{\varepsilon})$.

Schémas d'approximation polynomiaux efficaces

Un problème admet un **schéma d'approximation polynomial** si pour tout ε il admet un algorithme ε -approché de complexité polynomiale à ε fixé.

La classe de problème correspondante s'appelle **PTAS**.

Schémas d'approximation polynomiaux efficaces

Un problème admet un **schéma d'approximation polynomial** si pour tout ε il admet un algorithme ε -approché de complexité polynomiale à ε fixé.

La classe de problème correspondante s'appelle **PTAS**.

Le schéma d'approximation polynomial est **efficace** si la complexité est $O(n^c)$ pour une constante $c > 0$ (*i.e.* le degré ne dépend pas de ε).

La classe de problème correspondante s'appelle **EPTAS**.

Schémas d'approximation polynomiaux efficaces

Un problème admet un **schéma d'approximation polynomial** si pour tout ε il admet un algorithme ε -approché de complexité polynomiale à ε fixé.

La classe de problème correspondante s'appelle **PTAS**.

Le schéma d'approximation polynomial est **efficace** si la complexité est $O(n^c)$ pour une constante $c > 0$ (*i.e.* le degré ne dépend pas de ε).

La classe de problème correspondante s'appelle **EPTAS**.

Le schéma d'approximation est **totalelement polynomial** si la complexité est polynomiale en la taille et en ε .

La classe de problème correspondante s'appelle **FPTAS**.

Schémas d'approximation polynomiaux efficaces

Un problème admet un **schéma d'approximation polynomial** si pour tout ε il admet un algorithme ε -approché de complexité polynomiale à ε fixé.

La classe de problème correspondante s'appelle **PTAS**.

Le schéma d'approximation polynomial est **efficace** si la complexité est $O(n^c)$ pour une constante $c > 0$ (*i.e.* le degré ne dépend pas de ε).

La classe de problème correspondante s'appelle **EPTAS**.

Le schéma d'approximation est **totalelement polynomial** si la complexité est polynomiale en la taille et en ε .

La classe de problème correspondante s'appelle **FPTAS**.

Pour sac-à-dos, on a obtenu $O(n^2/\varepsilon)$: sac-à-dos est dans FPTAS.

Schémas d'approximation polynomiaux efficaces

Un problème admet un **schéma d'approximation polynomial** si pour tout ε il admet un algorithme ε -approché de complexité polynomiale à ε fixé.

La classe de problème correspondante s'appelle **PTAS**.

Le schéma d'approximation polynomial est **efficace** si la complexité est $O(n^c)$ pour une constante $c > 0$ (*i.e.* le degré ne dépend pas de ε).

La classe de problème correspondante s'appelle **EPTAS**.

Le schéma d'approximation est **totalement polynomial** si la complexité est polynomiale en la taille et en ε .

La classe de problème correspondante s'appelle **FPTAS**.

Pour sac-à-dos, on a obtenu $O(n^2/\varepsilon)$: sac-à-dos est dans FPTAS.

Par construction, **FPTAS** \subseteq **PTAS** \subseteq **APX**.

Cours 5: Algorithmes approchés

Algorithmes approchés

Schémas d'approximation

Bin-packing: limite de l'approximabilité.

Voyageur de commerce : non approximabilité

Rangement optimal et approximation

Donnée: n objets de poids p_1, \dots, p_n , et des boites de capacité P .

Problème: Mettre les objets dans un nombre minimum de boites.

Exemple: 61, 41, 40, 40, 20, 19, 19 $P = 120$

avec 2 boites: $61+40+19=120$ $41+40+20+19=120$.

Rangement optimal et approximation

Donnée: n objets de poids p_1, \dots, p_n , et des boites de capacité P .

Problème: Mettre les objets dans un nombre minimum de boites.

Exemple: 61, 41, 40, 40, 20, 19, 19 $P = 120$

avec 2 boites: $61+40+19=120$ $41+40+20+19=120$.

Algorithme glouton: ordonner les objets $p_1 \geq p_2 \geq \dots \geq p_n$, et les ranger dans cet ordre dans la première boite possible:

Rangement optimal et approximation

Donnée: n objets de poids p_1, \dots, p_n , et des boites de capacité P .

Problème: Mettre les objets dans un nombre minimum de boites.

Exemple: 61, 41, 40, 40, 20, 19, 19 $P = 120$

avec 2 boites: $61+40+19=120$ $41+40+20+19=120$.

Algorithme glouton: ordonner les objets $p_1 \geq p_2 \geq \dots \geq p_n$, et les ranger dans cet ordre dans la première boite possible:

Exemple: 61

Rangement optimal et approximation

Donnée: n objets de poids p_1, \dots, p_n , et des boites de capacité P .

Problème: Mettre les objets dans un nombre minimum de boites.

Exemple: 61, 41, 40, 40, 20, 19, 19 $P = 120$

avec 2 boites: $61+40+19=120$ $41+40+20+19=120$.

Algorithme glouton: ordonner les objets $p_1 \geq p_2 \geq \dots \geq p_n$, et les ranger dans cet ordre dans la première boite possible:

Exemple: 61 + 41

Rangement optimal et approximation

Donnée: n objets de poids p_1, \dots, p_n , et des boites de capacité P .

Problème: Mettre les objets dans un nombre minimum de boites.

Exemple: 61, 41, 40, 40, 20, 19, 19 $P = 120$

avec 2 boites: $61+40+19=120$ $41+40+20+19=120$.

Algorithme glouton: ordonner les objets $p_1 \geq p_2 \geq \dots \geq p_n$, et les ranger dans cet ordre dans la première boite possible:

Exemple: 61 + 41
 40

Rangement optimal et approximation

Donnée: n objets de poids p_1, \dots, p_n , et des boites de capacité P .

Problème: Mettre les objets dans un nombre minimum de boites.

Exemple: 61, 41, 40, 40, 20, 19, 19 $P = 120$

avec 2 boites: $61+40+19=120$ $41+40+20+19=120$.

Algorithme glouton: ordonner les objets $p_1 \geq p_2 \geq \dots \geq p_n$, et les ranger dans cet ordre dans la première boite possible:

Exemple: 61 + 41
 40 + 40

Rangement optimal et approximation

Donnée: n objets de poids p_1, \dots, p_n , et des boites de capacité P .

Problème: Mettre les objets dans un nombre minimum de boites.

Exemple: 61, 41, 40, 40, 20, 19, 19 $P = 120$

avec 2 boites: $61+40+19=120$ $41+40+20+19=120$.

Algorithme glouton: ordonner les objets $p_1 \geq p_2 \geq \dots \geq p_n$, et les ranger dans cet ordre dans la première boite possible:

Exemple: 61 + 41
 40 + 40 + 20

Rangement optimal et approximation

Donnée: n objets de poids p_1, \dots, p_n , et des boites de capacité P .

Problème: Mettre les objets dans un nombre minimum de boites.

Exemple: 61, 41, 40, 40, 20, 19, 19 $P = 120$

avec 2 boites: $61+40+19=120$ $41+40+20+19=120$.

Algorithme glouton: ordonner les objets $p_1 \geq p_2 \geq \dots \geq p_n$, et les ranger dans cet ordre dans la première boite possible:

Exemple:
61 + 41
40 + 40 + 20 + 19

Rangement optimal et approximation

Donnée: n objets de poids p_1, \dots, p_n , et des boites de capacité P .

Problème: Mettre les objets dans un nombre minimum de boites.

Exemple: 61, 41, 40, 40, 20, 19, 19 $P = 120$

avec 2 boites: $61+40+19=120$ $41+40+20+19=120$.

Algorithme glouton: ordonner les objets $p_1 \geq p_2 \geq \dots \geq p_n$, et les ranger dans cet ordre dans la première boite possible:

Exemple:

61	+	41				
40	+	40	+	20	+	19
19						

Rangement optimal et approximation

Donnée: n objets de poids p_1, \dots, p_n , et des boites de capacité P .

Problème: Mettre les objets dans un nombre minimum de boites.

Exemple: 61, 41, 40, 40, 20, 19, 19 $P = 120$

avec 2 boites: $61+40+19=120$ $41+40+20+19=120$.

Algorithme glouton: ordonner les objets $p_1 \geq p_2 \geq \dots \geq p_n$, et les ranger dans cet ordre dans la première boite possible:

Exemple:

61	+	41				
40	+	40	+	20	+	19
19						

Théorème: L'algorithme glouton utilise au plus $1 + \frac{3}{2}N_{\text{opt}}$ boites.

Rangement optimal et approximation

Donnée: n objets de poids p_1, \dots, p_n , et des boites de capacité P .

Problème: Mettre les objets dans un nombre minimum de boites.

Exemple: 61, 41, 40, 40, 20, 19, 19 $P = 120$

avec 2 boites: $61+40+19=120$ $41+40+20+19=120$.

Algorithme glouton: ordonner les objets $p_1 \geq p_2 \geq \dots \geq p_n$, et les ranger dans cet ordre dans la première boite possible:

Exemple:
61 + 41
40 + 40 + 20 + 19
19

Théorème: L'algorithme glouton utilise au plus $1 + \frac{3}{2}N_{\text{opt}}$ boites.

Preuve: • Si tous les objets ont poids $\geq P/3$, alors glouton est optimum:

Rangement optimal et approximation

Donnée: n objets de poids p_1, \dots, p_n , et des boites de capacité P .

Problème: Mettre les objets dans un nombre minimum de boites.

Exemple: 61, 41, 40, 40, 20, 19, 19 $P = 120$

avec 2 boites: $61+40+19=120$ $41+40+20+19=120$.

Algorithme glouton: ordonner les objets $p_1 \geq p_2 \geq \dots \geq p_n$, et les ranger dans cet ordre dans la première boite possible:

Exemple:
61 + 41
40 + 40 + 20 + 19
19

Théorème: L'algorithme glouton utilise au plus $1 + \frac{3}{2}N_{\text{opt}}$ boites.

Preuve: • Si tous les objets ont poids $\geq P/3$, alors glouton est optimum:
 $p_1 \geq \dots \geq p_i > \frac{2}{3}P \geq p_{i+1} \geq \dots \geq p_j > \frac{1}{2}P \geq p_{j+1} \geq \dots \geq p_n > \frac{1}{3}P$

Rangement optimal et approximation

Donnée: n objets de poids p_1, \dots, p_n , et des boites de capacité P .

Problème: Mettre les objets dans un nombre minimum de boites.

Exemple: 61, 41, 40, 40, 20, 19, 19 $P = 120$

avec 2 boites: $61+40+19=120$ $41+40+20+19=120$.

Algorithme glouton: ordonner les objets $p_1 \geq p_2 \geq \dots \geq p_n$, et les ranger dans cet ordre dans la première boite possible:

Exemple:
61 + 41
40 + 40 + 20 + 19
19

Théorème: L'algorithme glouton utilise au plus $1 + \frac{3}{2}N_{\text{opt}}$ boites.

Preuve: • Si tous les objets ont poids $\geq P/3$, alors glouton est optimum:

$$p_1 \geq \dots \geq p_i > \frac{2}{3}P \geq p_{i+1} \geq \dots \geq p_j > \frac{1}{2}P \geq p_{j+1} \geq \dots \geq p_n > \frac{1}{3}P$$

• Si chaque boite contient un objet de poids $> \frac{1}{3}P$ alors glouton optimum.

Rangement optimal et approximation

Donnée: n objets de poids p_1, \dots, p_n , et des boites de capacité P .

Problème: Mettre les objets dans un nombre minimum de boites.

Exemple: 61, 41, 40, 40, 20, 19, 19 $P = 120$

avec 2 boites: $61+40+19=120$ $41+40+20+19=120$.

Algorithme glouton: ordonner les objets $p_1 \geq p_2 \geq \dots \geq p_n$, et les ranger dans cet ordre dans la première boite possible:

Exemple: 61 + 41
40 + 40 + 20 + 19
19

Théorème: L'algorithme glouton utilise au plus $1 + \frac{3}{2}N_{\text{opt}}$ boites.

Preuve: • Si tous les objets ont poids $\geq P/3$, alors glouton est optimum:

$$p_1 \geq \dots \geq p_i > \frac{2}{3}P \geq p_{i+1} \geq \dots \geq p_j > \frac{1}{2}P \geq p_{j+1} \geq \dots \geq p_n > \frac{1}{3}P$$

• Si chaque boite contient un objet de poids $> \frac{1}{3}P$ alors glouton optimum.

• Sinon toutes les boites sauf une sont remplies à plus des $2/3$.

Rangement optimal et limite de l'approximabilité

Théorème: algorithme polynomial $\varepsilon < \frac{1}{3}$ approché pour RANGE OPT
 \Rightarrow algorithme polynomial pour PARTITION $\Rightarrow \mathcal{P} = \mathcal{NP}$.

Rangement optimal et limite de l'approximabilité

Théorème: algorithme polynomial $\varepsilon < \frac{1}{3}$ approché pour RANGE OPT
 \Rightarrow algorithme polynomial pour PARTITION $\Rightarrow \mathcal{P} = \mathcal{NP}$.

Problème PARTITION: trouver une partition S_1, S_2 d'un ensemble fini d'entiers S telle que $\sum_{s \in S_1} s = \sum_{s \in S_2} s = S/2$.

Rangement optimal et limite de l'approximabilité

Théorème: algorithme polynomial $\varepsilon < \frac{1}{3}$ approché pour RANGE OPT
 \Rightarrow algorithme polynomial pour PARTITION $\Rightarrow \mathcal{P} = \mathcal{NP}$.

Problème PARTITION: trouver une partition S_1, S_2 d'un ensemble fini d'entiers S telle que $\sum_{s \in S_1} s = \sum_{s \in S_2} s = S/2$.

Preuve que PARTITION est \mathcal{NP} -complet: clairement dans \mathcal{NP}

Réduction de SOMMEPARTIELLE à PARTITION.

Rangement optimal et limite de l'approximabilité

Théorème: algorithme polynomial $\varepsilon < \frac{1}{3}$ approché pour RANGE OPT
 \Rightarrow algorithme polynomial pour PARTITION $\Rightarrow \mathcal{P} = \mathcal{NP}$.

Problème PARTITION: trouver une partition S_1, S_2 d'un ensemble fini d'entiers S telle que $\sum_{s \in S_1} s = \sum_{s \in S_2} s = S/2$.

Preuve que PARTITION est \mathcal{NP} -complet: clairement dans \mathcal{NP}

Réduction de SOMMEPARTIELLE à PARTITION.

Preuve du théorème: $E = \{e_1, e_2, \dots, e_n\}$ une instance de PARTITION

On considère le problème RANGE OPT avec $P = \frac{1}{2} \sum e_i$

et on suppose d'y appliquer un algorithme $\varepsilon < \frac{1}{3}$ approché.

- si la réponse est 2, il y a une solution à PARTITION
- sinon, la réponse est 3 ou plus et il n'y a pas de solution avec 2 boîtes (car $\frac{3-2}{3} > \varepsilon$), et donc pas de solution à PARTITION

Rangement optimal et limite de l'approximabilité

Théorème: algorithme polynomial $\varepsilon < \frac{1}{3}$ approché pour RANGE OPT
 \Rightarrow algorithme polynomial pour PARTITION $\Rightarrow \mathcal{P} = \mathcal{NP}$.

Problème PARTITION: trouver une partition S_1, S_2 d'un ensemble fini d'entiers S telle que $\sum_{s \in S_1} s = \sum_{s \in S_2} s = S/2$.

Preuve que PARTITION est \mathcal{NP} -complet: clairement dans \mathcal{NP}

Réduction de SOMMEPARTIELLE à PARTITION.

Preuve du théorème: $E = \{e_1, e_2, \dots, e_n\}$ une instance de PARTITION

On considère le problème RANGE OPT avec $P = \frac{1}{2} \sum e_i$

et on suppose d'y appliquer un algorithme $\varepsilon < \frac{1}{3}$ approché.

- si la réponse est 2, il y a une solution à PARTITION
- sinon, la réponse est 3 ou plus et il n'y a pas de solution avec 2 boîtes (car $\frac{3-2}{3} > \varepsilon$), et donc pas de solution à PARTITION

Conclusion: PTAS \subsetneq APX

Cours 5: Algorithmes approchés

Algorithmes approchés

Schémas d'approximation

Bin-packing: limite de l'approximabilité.

Voyageur de commerce : non approximabilité

Voyageur du commerce et inapproximabilité

Donnée: un graphe G et une valuation des arêtes ("distances entre villes")

Problème: Trouver un ordre de visite des villes qui minimise la distance.

Théorème: algo polynomial $\varepsilon < 1$ approché pour le voyageur de commerce
 \Rightarrow algorithme polynomial pour HAMILTONIEN $\Rightarrow \mathcal{P} = \mathcal{NP}$.

Voyageur du commerce et inapproximabilité

Donnée: un graphe G et une valuation des arêtes ("distances entre villes")

Problème: Trouver un ordre de visite des villes qui minimise la distance.

Théorème: algo polynomial $\varepsilon < 1$ approché pour le voyageur de commerce

\Rightarrow algorithme polynomial pour HAMILTONIEN $\Rightarrow \mathcal{P} = \mathcal{NP}$.

Problème HAMILTONIEN: étant donné un graphe, trouver un cycle qui visite une et une seule fois les sommets.

Voyageur du commerce et inapproximabilité

Donnée: un graphe G et une valuation des arêtes ("distances entre villes")

Problème: Trouver un ordre de visite des villes qui minimise la distance.

Théorème: algo polynomial $\varepsilon < 1$ approché pour le voyageur de commerce

\Rightarrow algorithme polynomial pour HAMILTONIEN $\Rightarrow \mathcal{P} = \mathcal{NP}$.

Problème HAMILTONIEN: étant donné un graphe, trouver un cycle qui visite une et une seule fois les sommets. \mathcal{NP} -complet (cf poly)

Voyageur du commerce et inapproximabilité

Donnée: un graphe G et une valuation des arêtes ("distances entre villes")

Problème: Trouver un ordre de visite des villes qui minimise la distance.

Théorème: algo polynomial $\varepsilon < 1$ approché pour le voyageur de commerce

\Rightarrow algorithme polynomial pour HAMILTONIEN $\Rightarrow \mathcal{P} = \mathcal{NP}$.

Problème HAMILTONIEN: étant donné un graphe, trouver un cycle qui visite une et une seule fois les sommets. \mathcal{NP} -complet (cf poly)

Preuve du théorème: soit G un graphe (instance de HAMILTONIEN)

Voyageur du commerce et inapproximabilité

Donnée: un graphe G et une valuation des arêtes ("distances entre villes")

Problème: Trouver un ordre de visite des villes qui minimise la distance.

Théorème: algo polynomial $\varepsilon < 1$ approché pour le voyageur de commerce

\Rightarrow algorithme polynomial pour HAMILTONIEN $\Rightarrow \mathcal{P} = \mathcal{NP}$.

Problème HAMILTONIEN: étant donné un graphe, trouver un cycle qui visite une et une seule fois les sommets. \mathcal{NP} -complet (cf poly)

Preuve du théorème: soit G un graphe (instance de HAMILTONIEN)

• On pose k entier $\geq \frac{1}{1-\varepsilon}$ et on constuit la valuation:

$$v(x, y) = 1 \text{ si } (x, y) \text{ arête de } G, \quad v(x, y) = nk + 1 \text{ sinon.}$$

Voyageur du commerce et inapproximabilité

Donnée: un graphe G et une valuation des arêtes ("distances entre villes")

Problème: Trouver un ordre de visite des villes qui minimise la distance.

Théorème: algo polynomial $\varepsilon < 1$ approché pour le voyageur de commerce

\Rightarrow algorithme polynomial pour HAMILTONIEN $\Rightarrow \mathcal{P} = \mathcal{NP}$.

Problème HAMILTONIEN: étant donné un graphe, trouver un cycle qui visite une et une seule fois les sommets. \mathcal{NP} -complet (cf poly)

Preuve du théorème: soit G un graphe (instance de HAMILTONIEN)

• On pose k entier $\geq \frac{1}{1-\varepsilon}$ et on construit la valuation:

$$v(x, y) = 1 \text{ si } (x, y) \text{ arête de } G, \quad v(x, y) = nk + 1 \text{ sinon.}$$

• Si on a une ε -approx de valuation n alors G admet un cycle hamiltonien, sinon la valuation trouvée est $x \geq (n-1) + (nk+1)$ et comme $\frac{n+nk-C_o}{n+nk} \leq \varepsilon$, on a $C_o \geq n(1+k)(1-\varepsilon) > n$.

Voyageur du commerce et inapproximabilité

On vient de voir que $TSP \notin APX$

Voyageur du commerce et inapproximabilité

On vient de voir que $TSP \notin APX$

On a vu en TD (exo 5 de la PC4) que par contre

$Euclidean\ TSP \in APX$ (facteur 2)

Voyageur du commerce et inapproximabilité

On vient de voir que $TSP \notin APX$

On a vu en TD (exo 5 de la PC4) que par contre

$Euclidean\ TSP \in APX$ (facteur 2)

En fait il existe un schéma d'approximation polynomial pour Euclidean TSP : $Euclidean\ TSP \in PTAS$.

Ce résultat date de 1998 et a valu le prix Gödel à ses auteurs en 2010.

Cours 5: Algorithmes approchés

Algorithmes approchés

Schémas d'approximation

Bin-packing: limite de l'approximabilité.

Voyageur de commerce : non approximabilité

Que retenir ?

Algorithme approché : chercher une borne sur l'optimum !

Limite de l'approximabilité : via la NP-complétude

Des résultats d'approximabilités très différents
pour des problèmes tous NP-complets !