Algorithmes gloutons

Stéphane Grandcolas

stephane.grandcolas@univ-amu.fr

Problèmes d'optimisation.

[Solutions]

- permutations, sous-ensembles, configurations particulières,. . .
- *problème de décision* : existe-t-il une solution ?

[Solutions optimales]

- les solutions qui maximisent (resp. minimisent) une fonction donnée
- *problème d'optimisation* : trouver une solution optimale.

Problèmes d'optimisation.

[Exemples]

- arbres couvrants de poids minimal,
- plus longue sous séquence commune,
- plus court trajet passant par une ensemble de villes donné,
- permutation d'une séquence de colis la mieux équilibrée,
- . . .

Problèmes d'optimisation.

[Approche générale]

- un premier choix,
- des sous-problèmes correspondants aux différents choix,
- une solution optimale est déduite des solutions optimales des sous-problèmes.

[Une solution est une suite de choix]

Exemple : placement des convives.

Placer n convives en minimisant le nombre de voisins en conflits.

placer_les_convives (E, P)

1 **si** $E = \emptyset$ **alors**
2 **si** P est meilleur que M **alors**
3 $M := P$,
4 **sinon**
5 **pour chaque** $c \in E$ **faire** *on envisage successivement de placer*
6 **placer_les_convives** $(E - \{c\}, P \oplus c)$, *chaque convive de E à la position coura*

On va énumérer les $n!$ permutations des n convives

Exemple : placement des convives.

Approche glouton : on envisage un seul choix à chaque étape, celui qui semble le meilleur à cet instant (par exemple le convive qui devrait être le plus difficile à placer).

placer_les_convives (E)

1 $S := \langle\rangle$,
2 **tant que** $E \neq \emptyset$ **faire**
3 $U := \{ c \in E$ tel que c n'est pas en conflit avec le dernier convive de $S\}$,
4 **si** $U = \emptyset$ **alors**
5 $U := E$,
6 choisir le convive $c \in U$ qui est le plus en conflit avec des convives de E,
7 $E := E - \{c\}$,
8 $S := S \oplus c$,
9 **renvoyer** S

Pas de garantie de l'optimalité mais coût réduit.

Algorithmes gloutons.

recherche exhaustive : on explore systématiquement tous les choix possibles,

programmation dynamique : mémorisation des solutions de sous-problèmes qui apparaissent plusieurs fois (si il y en a),

algorithmes gloutons : sélection d'un choix particulier et traitement du sous-problème correspondant (et non pas de tous les sous-problèmes) : *heuristique de choix*

Algorithmes gloutons.

Un algorithme glouton construit une solution pas à pas

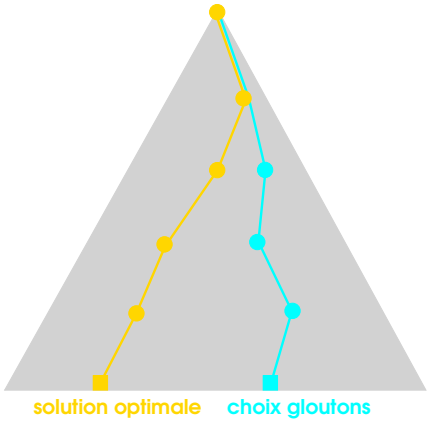
- **sans revenir sur ses décisions,**
- **en effectuant à chaque étape le choix qui semble le meilleur,**
- **en espérant obtenir un résultat optimum global**

Exemple rendu de monnaie : avec le moins possible de pièces.

Algorithme glouton : répéter le choix de la pièce de plus grande valeur qui ne dépasse pas la somme restante tant que celle-ci n'est pas nulle

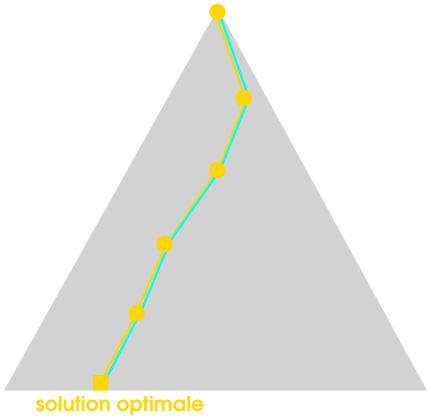
Algorithmes gloutons.

Algorithmes gloutons.



Choix glouton : solution non optimale (*heuristique glouton*)

Algorithmes gloutons.



Choix glouton optimal

Algorithmes gloutons.

Approche glouton

- suivant les problèmes pas de garantie d'optimalité (heuristique glouton)
- peu couteuse (comparée à une énumération exhaustive)
- choix *intuitif*

Une très bonne approche lorsqu'elle produit des solutions optimales

Une bonne approche pour des problèmes difficiles (i.e. les autres approches sont très couteuses) où on ne cherche pas absolument l'optimalité

Exemple : empaquetage.

un ensemble d'objets $E = \{1, \ldots, n\}$ de poids $p_1, \ldots, p_n$

des boites de capacités C

Problème : placer les objets dans les boites

- en respectant leurs capacités
- en utilisant le moins possible de boites

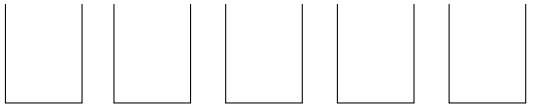
Méthode : par choix successifs d'un objet et d'une boite

Choix glouton : placer le premier objet dans la première boîte où c'est possible

Exemple : empaquetage.

Exemple.

poids 7 6 3 4 8 5 9 2

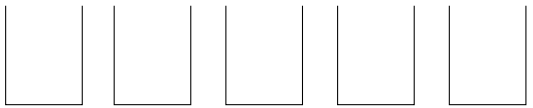


capacité 11

Exemple : empaquetage.

Exemple.

poids 7 6 3 4 8 5 9 2

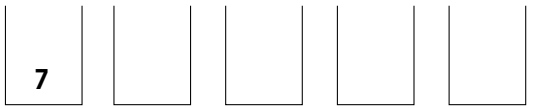


capacité 11

Exemple : empaquetage.

Exemple.

poids 7 6 3 4 8 5 9 2

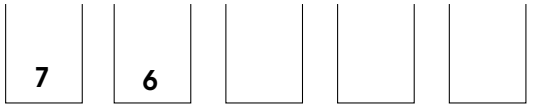


capacité 11

Exemple : empaquetage.

Exemple.

poids 7 6 3 4 8 5 9 2

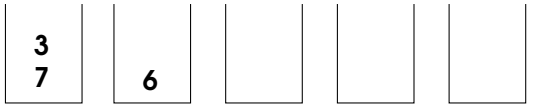


capacité 11

Exemple : empaquetage.

Exemple.

poids 7 6 3 4 8 5 9 2

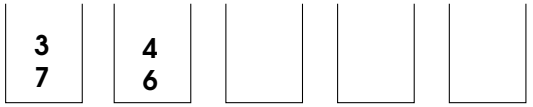


capacité 11

Exemple : empaquetage.

Exemple.

poids 7 6 3 4 8 5 9 2

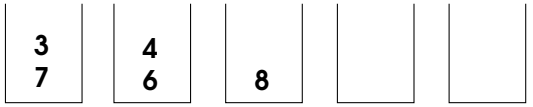


capacité 11

Exemple : empaquetage.

Exemple.

poids 7 6 3 4 8 5 9 2

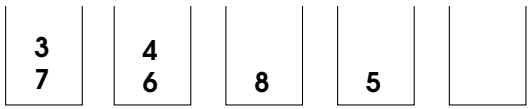


capacité 11

Exemple : empaquetage.

Exemple.

poids 7 6 3 4 8 5 9 2

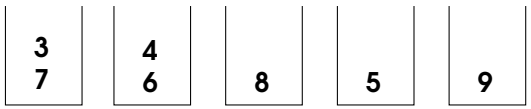


capacité 11

Exemple : empaquetage.

Exemple.

poids 7 6 3 4 8 5 9 2

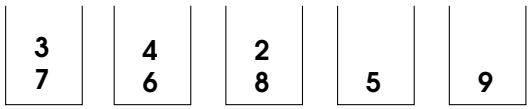


capacité 11

Exemple : empaquetage.

Exemple.

poids 7 6 3 4 8 5 9 2



capacité 11

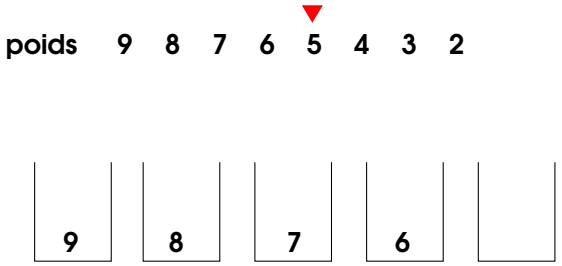
Exemple : empaquetage.

procédure **Empaqueter**(n, p, C)
n objets, $p[\,]$ leurs poids ($p[i] \leq C$),
C la capacité des boites,
out : le nombre de boites utilisées,
$c[\,]$ représente les capacités résiduelles des boites

```
1   m = 0,
2   c[1] = C,
3   pour i = 1 à n faire
4       b = 1,
5       tant que p[i] > c[b] et b ≤ m faire
6           b = b + 1,
7       si b = m + 1 alors
8           m = m + 1,
9           c[b] = C,
10      c[b] = c[b] − p[i],
11  renvoyer m
```

Exemple : empaquetage.

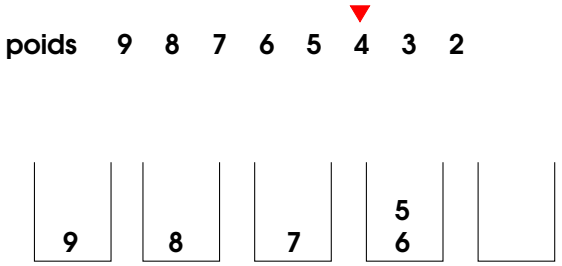
Un bon choix glouton : prendre l'objet de plus grand poids



capacité 11

Exemple : empaquetage.

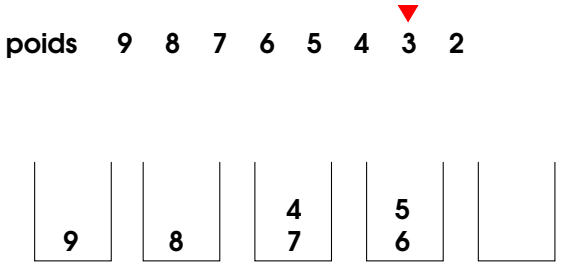
Un bon choix glouton : prendre l'objet de plus grand poids



capacité 11

Exemple : empaquetage.

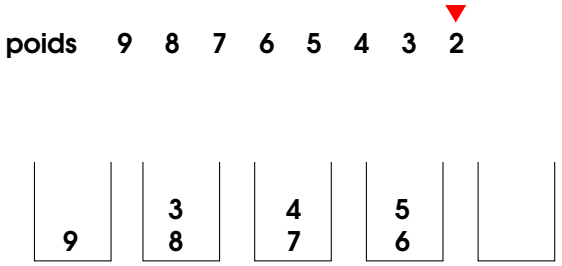
Un bon choix glouton : prendre l'objet de plus grand poids



capacité 11

Exemple : empaquetage.

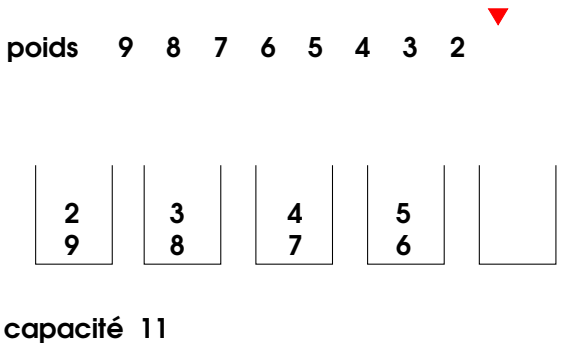
Un bon choix glouton : prendre l'objet de plus grand poids



capacité 11

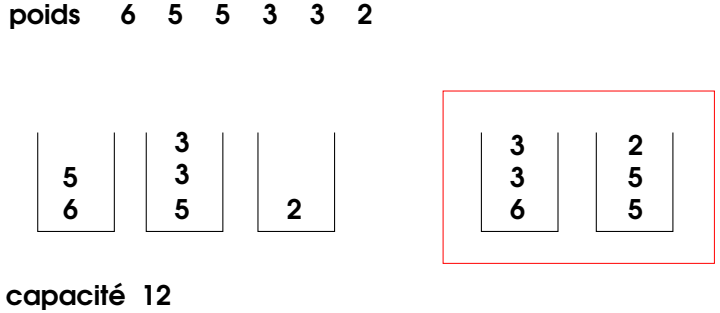
Exemple : empaquetage.

Un bon choix glouton : prendre l'objet de plus grand poids



Exemple : empaquetage.

Le choix du paquet de plus grand poids n'est pourtant pas optimal



Algorithmes gloutons : optimalité.

Un algorithme glouton produit des solutions optimales si les deux propriétés suivantes sont vérifiées :

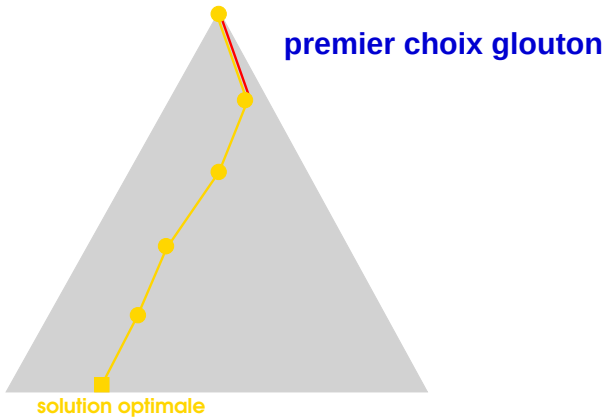
[propriété du choix glouton] il existe toujours une solution optimale qui contient un premier choix glouton

- En général on montre que toute solution optimale contient ou débute par un choix glouton.

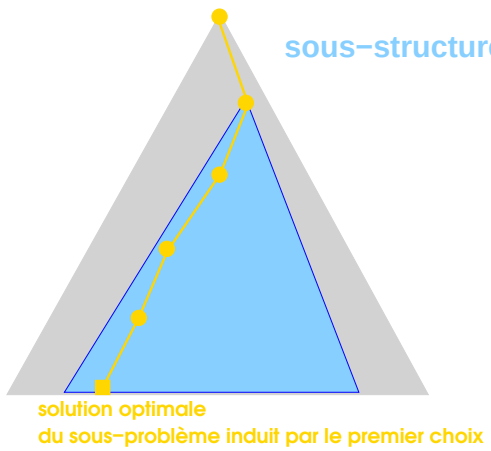
[propriété de sous-structure optimale] toute solution optimale contient une sous-structure optimale

- Soit S une solution optimale du problème P contenant le choix $\mathcal{C}$, et $S' = S \setminus \{\mathcal{C}\}$.
  Alors, S' est une solution optimale du sous-problème $P_{\mathcal{C}}$ résultant du choix $\mathcal{C}$ dans le problème P.

Algorithmes gloutons : choix optimal.



Algorithmes gloutons : choix optimal.



Exemple : rendu de monnaie.

s : somme à atteindre
M : valeurs des pièces de monnaie

Objectif : constituer la somme s avec le moins possible de pièces

Exemple : $M = \{1, 2, 5\}$ et $s = 10$
Solution optimale : $5 + 5$ (2 pièces)

Exemple : rendu de monnaie.

Algorithme glouton : choisir des pièces tant que la somme n'est pas atteinte

Heuristique de choix : la pièce de valeur la plus grande (inférieure à la somme restant à compléter)

Rendu de monnaie.

[propriété du choix glouton]

Remarque : toute solution optimale contient deux pièces de 2 et aucune de 1 ou au plus une pièce de 1 et au plus une pièce de 2

$$0/0 \quad 0/1 \quad 1/0 \quad 1/1 \quad 2/0$$

Si $s \geq 5$ toute solution optimale contient au moins une pièce de 5 : c'est le choix glouton.

Si s vaut 1, 2, 3 ou 4, l'algorithme fait un bon premier choix (la valeur 2 ou la valeur 1)

Toute solution optimale contient donc un choix glouton

Rendu de monnaie.

[propriété de sous-structure optimale]

Soit S une solution optimale pour la somme s,
soit $\mathcal{C} \in S$ un choix glouton (la plus grosse pièce de S).
Alors $S' = S - \{\mathcal{C}\}$ est une solution optimale pour $s - valeur(\mathcal{C})$.

Preuve. Par l'absurde : soit S'' meilleure que S' pour $s - valeur(\mathcal{C})$, alors $S'' \cup \{\mathcal{C}\}$ est meilleure que $S' \cup \{\mathcal{C}\}$ pour s.
Contredit l'hypothèse que $S' \cup \{\mathcal{C}\} = S$ est optimale.

Rendu de monnaie : preuve de l'optimalité.

$\mathcal{P}_n$: caractérise les problèmes dont les solutions optimales comportent au plus n pièces

Preuve par récurrence sur n :

- **base :** le choix glouton produit des solutions optimales pour les problèmes $\mathcal{P}_1$ (la solution se réduit à la plus grande pièce de valeur inférieure ou ègale à s)
- **hypothèse :** l'algorithme glouton produit des solutions optimales pour les problèmes $\mathcal{P}_n$
  induction : l'algorithme glouton produit des solutions optimales pour les problèmes $\mathcal{P}_{n+1}$

Rendu de monnaie : preuve de l'optimalité.

preuve :

- soit S une solution optimale de taille $n+1$ (somme s),
- S contient donc un premier choix glouton $\mathcal{C} \in S$ (**propriété du choix glouton**).

Alors $S \setminus \{\mathcal{C}\}$ est une solution optimale de taille n pour $s - valeur(\mathcal{C})$ (**propriété de sous-structure optimale**).

Hypothèse de récurrence : l'algorithme glouton produit une solution optimale S' pour la somme $s - valeur(\mathcal{C})$.

$S' \cup \{\mathcal{C}\}$ constitue donc une solution optimale pour s correspondant au choix glouton.