Algorithmes gloutons [gl]
Algorithmique

Karine Zampieri, Stéphane Rivière, Béatrice Amerein-Soltner

Unisciel algoprog Université Haute-Alsace

Version 21 mai 2018

Table des matières

Algorithmes gloutons

Mots-Clés Techniques de conception, Algorithmes gloutons ■
Requis Axiomatique impérative, Récursivité des actions, Complexité des algorithmes ■
Difficulté ● ● ○

Objectif
Ce module présente le paradigme de **l'algorithme glouton** puis l'applique à plusieurs exemples. Les dernières sections donnent les éléments de la stratégie gloutonne ainsi que les fondements théoriques des méthodes gloutonnes.

1 Paradigme de l'algorithme glouton

Dans ce module, on se concentre sur les problèmes d'optimisation.

Algorithme glouton (*greedy* en anglais)
(Dit aussi **vorace**) Procédure algorithmique qui construit une solution d'une manière incrémentale. A chaque étape, cette technique prend la direction la plus prometteuse, suivant des règles bien simples, en considérant une seule donnée à la fois.

Globalement optimale ou heuristique

Ce choix, localement optimal, n'a aucune raison de conduire à une solution globalement optimale. Cependant, certains problèmes peuvent être résolus d'une manière optimale ainsi. Autrement dit, **l'optimalité locale conduit à l'optimalité globale**.

Dans d'autres cas, la méthode gloutonne est seulement une heuristique qui ne conduit qu'à une **solution sous-optimale**, mais qui peut être utilisée quand on ne connaît pas d'algorithme exact efficace. Le module @[Heuristiques] revient sur les algorithmes voraces en tant que solutions heuristiques.

Algorithme générique d'une procédure gloutonne
Il peut être résumé comme suit :

```
Algorithme vorace
Début
S <- EnsembleVide
C <- ensemble des candidats à la solution
Tant que S n'est pas une solution et C <> EnsembleVide Faire
| x <- choisir un élément de C le plus prometteur
| C <- C - x
| Si realisable(solution,x) Alors
| | solution <- union(solution, x)
| Finsi
FinTantQue
Si S est une solution Alors
| Retourner S
Sinon
| Retourner pas de solution
FinSi
Fin
```

Le choix de la donnée la plus prometteuse, lors de la construction de la solution, se fait suivant une règle. Cette règle définit en réalité la stratégie de l'algorithme glouton ainsi conçu.

Remarque
A la fin de leur exécution, les algorithmes gloutons ne génèrent qu'une seule solution. De plus, un algorithme glouton ne revient jamais sur une décision prise précédemment.

Remarque
La fonction de sélection est généralement basée sur la fonction objectif à optimiser.

2 Exemple : Rendu de la monnaie

Afin de clarifier la terminologie introduite ci-avant :

Problème
Soit le problème de rendre la monnaie à un client en lui donnant le moins de pièces possibles.

Éléments du schéma
Ils sont résumés comme suit :

1. Les candidats potentiels à la solution : ensemble fini de pièces de monnaie, par exemple : 1, 5, 10 et 25 centimes.
2. Une solution : le total de l'ensemble de pièces choisies correspondant exactement au montant à payer.
3. Une solution réalisable : le total de l'ensemble de pièces choisies ne dépasse pas le montant à payer.
4. La donnée la plus prometteuse est la grande pièce qui reste dans l'ensemble des candidats.
5. Fonction objectif : minimiser le nombre de pièces utilisées dans la solution.

Algorithme
L'algorithme glouton pour cet exemple est :

```
Algorithme vorace
Début
S <- EnsembleVide
C <- ensemble des candidats à la solution
TantQue total(S) < somme à payer et C <> EnsembleVide Faire
| x <- plus grande pièce
| C <- C - x
| Si total (S ) + x < somme à payer Alors
| | solution <- union(solution, x)
| Finsi
FinTantQue
Si S est une solution Alors
| retourner S
Sinon
| retourner pas de solution
FinSi
Fin
```

Exemple
Soit la monnaie de 87 centimes à rendre en utilisant seulement {25, 10, 5, 1}. L'algorithme ci-dessus va générer la solution : 3*25 + 1*10 + 2*1.

3 Exemple : Location d'une voiture

3.1 Problème

On considère le problème de la location d'une unique voiture. Des clients formulent un ensemble de demandes de location avec, pour chaque demande, le jour du début de la location et le jour de restitution du véhicule. Le problème est d'affecter le véhicule de manière à satisfaire le **maximum de clients** possible (et non pas de maximiser la somme des durées des locations).

On dispose donc d'un ensemble E des demandes de location avec, pour chaque élément e de E, la date $d(e)$ du début de la location et la date $f(e)$ de la fin de cette location. On veut obtenir un ensemble F maximal de demandes satisfaites.

Cet ensemble F doit vérifier une unique contrainte : deux demandes ne doivent pas se chevaucher dans le temps, c.-à-d. une location doit se terminer avant que la suivante ne commence. Cette contrainte s'écrit mathématiquement :

$$\forall e_1 \in F, \forall e_2 \in F : d(e_1) \leq d(e_2) \Rightarrow f(e_1) \leq d(e_2)$$

3.2 Algorithme

Algorithme

```
Fonction locationDUneVoiture(E)
Début
| Tri des éléments de E par date de fin croissante:
    On obtient une suite e[1],e[2]...e[n]
    telle que f(e[1])<=f(e[2])<=...<=f(e[n])
| F[1] <- e[1]
| j <- 1
| Pour k <- 1 à n Faire
| | Si d(e[k]) >= f(F[j]) Alors
| | | j <- j + 1
| | | F[j] <- e[j]
| | FinSi
| FinPour
| Retourner (F)
Fin
```

Complexité

Cet algorithme est glouton car à chaque étape il prend la location « la moins coûteuse » : celle qui finit le plus tôt parmi celles qui sont satisfiables.

3.3 Preuve de l'optimalité de l'algorithme

Soit $F = \{x_1, x_2, ..., x_p\}$ la solution obtenue par l'algorithme glouton, et soit $G = \{y_1, y_2, ..., y_q\}$, $q \geq p$, une solution optimale. On veut montrer que F est optimal et donc que $q = p$.

Supposons que les ensembles F et G sont classés par dates de fin de location croissantes. Si G (solution optimale) ne contient pas F (une solution obtenue), il existe un entier k tel que :

$$\forall i < k, x_i = y_i \text{ et } x_k \neq y_k$$

Par construction de F, x_k est une demande de location qui a la date de fin minimale et dont la date de début est postérieure à la date de fin de $x_{k-1} = y_{k-1}$. Par conséquent,

$$f(y_k) \geq f(x_k)$$

On peut alors remplacer G par $G' = \{y_1, y_2, ..., y_{k-1}, x_k, y_{k+1}, ..., y_q\}$ tout en satisfaisant la contrainte de non chevauchement des demandes. G' est une autre solution optimale mais ayant strictement plus d'éléments en commun avec F que G. En répétant autant que faire se peut ce procédé, on obtient un ensemble H de même cardinalité que G et qui contient F. Cet ensemble H ne peut contenir d'autres éléments que ceux de F car ceux-ci, débutants après la fin de x_p, auraient été ajoutés à F par l'algorithme glouton. Donc $H = F$ et F et G ont le même nombre d'éléments.

3.4 Limites de l'algorithme

Il est primordial, ici, que les demandes soient classées par **dates de fin croissantes**. Le tableau suivant présente trois demandes de location classées par dates de début croissantes pour lesquelles l'algorithme glouton présenté ci-dessus n'est pas optimal.

	e_1	e_2	e_3
d	2	3	5
F	8	4	8

Pour d'évidentes raisons de symétrie, classer les demandes par dates de début décroissantes donne par contre un résultat optimal.

	e_1	e_2	e_3
d	5	3	2
F	6	7	5

Attention
L'algorithme glouton ne donne pas l'optimum si notre but est de maximiser la durée totale de location du véhicule. Même si on classe les demandes de location par durées décroissantes, un algorithme glouton ne donnera pas une solution optimale : le tableau ci-dessus présente un contre-exemple. En fait, le problème de la maximisation de cette durée totale est NP-complet (cf. module @[NP-complétude]) et on ne connaît pas d'algorithme de complexité polynomiale pour le résoudre.

Remarque
Si on dispose de deux voitures et non plus d'une seule, l'algorithme précédent ne donne plus l'optimum.

4 Éléments de la stratégie gloutonne

Un algorithme glouton détermine une solution après avoir effectué une série de choix. Pour chaque point de décision, il retient le choix qui semble le meilleur à cet instant. Cette stratégie ne produit pas toujours une solution optimale. Il existe cependant deux caractéristiques qui indiquent qu'un problème se prête à une stratégie gloutonne : la propriété du choix glouton et une sous-structure optimale.

Propriété du choix glouton

On peut arriver à une solution globalement optimale en effectuant un choix localement optimal (ou choix glouton). En programmation dynamique on fait un choix à chaque étape, mais ce choix dépend de la solution de sous-problèmes, au contraire, dans un algorithme glouton, on fait le choix qui semble le meilleur sur le moment *puis* on résout les sous-problèmes qui surviennent une fois le choix fait. Une stratégie gloutonne progresse en général de manière descendante en faisant se succéder les choix gloutons pour ramener itérativement chaque instance du problème à une instance « plus petite ».

Sous-structure optimale

Montrer qu'un choix glouton aboutit à un problème similaire mais « plus petit » ramène la démonstration de l'optimalité à prouver qu'une solution optimale doit faire apparaître une sous-structure optimale.

Un problème fait apparaître une **sous-structure optimale** si une solution optimale contient la solution optimale de sous-problèmes. Cette propriété est un indice important de l'applicabilité de la programmation dynamique comme des algorithmes gloutons.

5 Fondements théoriques des méthodes gloutonnes

La théorie des matroïdes ne couvre pas tous les cas d'applications de la méthode gloutonne, mais elle couvre de nombreux cas intéressants en pratique.

5.1 Matroïdes

Matroïde
Couple $M = (E, I)$ vérifiant les conditions suivantes :

1. E est un ensemble fini non vide.
2. I est une famille non vide de sous-ensembles de E, appelés sous-ensembles **indépendants** de E, telle que si $H \in I$ et si $F \subset H$ alors $F \in I$ (on dit que I est **héréditaire**). Autrement dit, si I contient un sous-ensemble H de E, I contient tous les sous-ensembles de H. On remarque que l'ensemble vide est obligatoirement membre de I.
3. Si F et H sont deux éléments de I, avec $|F| < |H|$, alors il existe (au moins) un élément $x \in H \setminus F$ tel que $F \cup \{x\} \in I$ (**propriété d'échange**).

Théorème
Tous les sous-ensembles indépendants maximaux d'un matroïde ont la même taille.

Preuve
Ce résultat est une conséquence directe de la propriété d'échange : si un de ces ensembles, H, est strictement plus petit que les autres, la propriété d'échange nous garantit que I contient un sur-ensemble strict H' de H, ce qui contredit la maximalité de H. ■

Matroïde pondéré
Un **matroïde** $M = (E, I)$ est dit **pondéré** si l'on dispose d'une **fonction de pondération** w qui affecte un poids strictement positif $w(x)$ à chaque élément x de E.

La fonction de pondération w s'étend aux sous-ensembles de E.
Soit F un sous-ensemble quelconque de E :

$$w(F) = \sum_{x \in F} w(x)$$

5.2 Algorithmes gloutons sur un matroïde pondéré

De nombreux problèmes pour lesquels une approche gloutonne donne les solutions optimales peuvent être ramenés à une recherche d'un sous-ensemble indépendant de pondération maximale dans un matroïde pondéré. Autrement dit, on dispose d'un matroïde pondéré $M = (E, I)$ et on souhaite trouver un ensemble indépendant $F \in I$ pour lequel $w(F)$ est maximisé. Un tel sous-ensemble indépendant et qui possède la plus grande

pondération possible est appelé sous-ensemble **optimal** du matroïde. Comme la pondération est strictement positive par définition, un sous-ensemble optimal est toujours un sous-ensemble indépendant maximal.

L'algorithme ci-dessous prend en entrée un matroïde pondéré $M = (E, I)$ et sa fonction de pondération w et retourne un sous-ensemble optimal F.

Algorithme

```
Algorithme Glouton(M=(E,I), w)
Début
| F <- EnsVide
| Trier E par ordre de poids décroissant
| Pour x dans E par ordre de poids décroissant Faire
| | Si union(F,x) dans E Alors
| | | F <- union(F,x)
| | Finsi
| FinPour
| Retourner F
Fin
```

Complexité

Cet algorithme est glouton parce qu'il considère les éléments de E par ordre de poids décroissant et qu'il ajoute immédiatement un élément x à F si $F \cup \{x\}$ est indépendant. Si E contient n éléments et si la vérification de l'indépendance de $F \cup \{x\}$ prend un temps $O(f(n))$, l'algorithme tout entier s'exécute en $O(n \log n + nf(n))$ — rappel : le tri d'un ensemble de n éléments coûte $O(n \log n)$.

Conclusion

Le sous-ensemble F de E est indépendant par construction.
Nous allons maintenant établir l'optimalité de F.

Théorème : Propriété du choix glouton

Soit $M = (E, I)$ un matroïde pondéré de fonction de pondération w. Supposons que E soit trié par ordre de poids décroissant. Soit x le premier élément de E tel que $\{x\}$ soit indépendant, s'il existe.

Si x existe, il existe un sous-ensemble optimal F de E contenant x.

Preuve

Si x n'existe pas, le seul élément de I est l'ensemble vide.

Soit H un sous-ensemble optimal. On utilise H pour construire, au moyen de la propriété d'échange, un ensemble F maximal (de même cardinalité que H) et contenant x. Par construction, F et H ne diffèrent que d'un élément et il existe donc un élément y tel que : $F = (H \setminus \{y\}) \cup \{x\}$. Par maximalité du poids de x, $w(y) \leq w(x)$, $w(H) \leq w(F)$ et F est optimal. ■

Théorème
Soit $M = (E, I)$ un matroïde quelconque. Si x est un élément de E tel que $\{x\}$ n'est pas élément de I, alors x n'appartient à aucun sous-ensemble indépendant F de E.

Preuve
Autrement dit, un élément qui n'est pas utilisable immédiatement ne pourra jamais être utilisé : l'algorithme `Glouton` ne fait donc pas d'erreur en ne considérant pas les éléments de E qui ne sont pas extension de $\emptyset$. ■

Théorème : Propriété de sous-structure optimale
Soit x le premier élément de E choisi par `Glouton` pour le matroïde pondéré $M = (E, I)$. Le reste du problème — trouver un sous-ensemble indépendant contenant x et de poids maximal — se réduit à trouver un sous-ensemble indépendant et de poids maximal du matroïde pondéré $M' = (E', I')$, où :

$$E' = \{y \in E : \{x, y\} \in I\},$$
$$I' = \{H \subset E \setminus \{x\} : H \cup \{x\} \in I\},$$

et où la fonction de pondération de M' est celle de M restreinte à E'.

Preuve
Une solution de poids maximum sur M contenant x engendre une solution de poids maximum sur M', et vice versa. ■

Théorème : Validité de l'algorithme glouton
Si $M = (E, I)$ est un matroïde pondéré de fonction de pondération w, alors l'appel `Glouton(M,w)` avec $M = (E, I)$ renvoie un sous-ensemble optimal.

6 Conclusion

Algorithme glouton

Les algorithmes qui résolvent les problèmes d'optimisation parcourent en général une série d'étapes, au cours desquelles ils sont confrontés à un ensemble d'options. Pour de nombreux problèmes d'optimisation la programmation dynamique est une approche trop lourde pour déterminer les meilleures solutions ; d'autres algorithmes plus simples et efficaces y arriveront. Un **algorithme glouton** fait toujours le choix qui semble le meilleur sur le moment. Autrement dit, il fait un choix optimal localement, dans l'espoir que ce choix mènera à la solution optimale globalement.

Avantages, Inconvénients

L'intérêt et la force des algorithmes gloutons résident dans leur simplicité de construire une solution. De plus, cette solution se construit généralement en des temps raisonnables. L'inconvénient de cette technique est de toujours recourir à une preuve pour montrer l'optimalité de la solution ainsi générée, si optimalité il y a.

Optimalité

Les algorithmes gloutons n'aboutissent pas toujours à des solutions optimales, mais la méthode gloutonne est très puissante et fonctionne correctement pour des problèmes variés.