Algorithmes gloutons

Irena.Rusu@univ-nantes.fr

LINA, bureau 123, 02.51.12.58.16

Sommaire

- Types de problèmes
- Résolution et algorithmes gloutons
- Approche intuitive des algorithmes gloutons
  - Principe
  - 1ère application : le problème du Sac à dos
- Variante standard et variations
  - 2ème application : le problème de l'affectation de ressources
  - Correction et preuve

Sommaire

- Types de problèmes
- Résolution et algorithmes gloutons
- Approche intuitive des algorithmes gloutons
  - Principe
  - 1ère application : le problème du Sac à dos
- Variante standard et variations
  - 2ème application : le problème de l'affectation de ressources
  - Correction et preuve

Types de problèmes

- Problème de décision
  - Entrée : Tableau de notes pour un groupe d'étudiants.
  - Question : Existe-t-il une note de 13 ?

  La réponse est toujours Oui/Non.

- Problème d'optimisation
  - Entrée : Tableau de notes pour un groupe d'étudiants.
  - Question : Quelle est la note la plus proche de 13 ?

  Il y a souvent plusieurs réponses (ex. 12.5 ou 13.5)

  Il y a un critère pour évaluer une réponse (ici, la proximité de 13).

Définition d’un problème d’optimisation

- Problème d’optimisation
  - Une entrée : ce qui est fourni
  - Une description du type de réponse recherché.
  - Un coût : une mesure de la qualité de la réponse.
  - Un but : maximiser ou minimiser.

- Solution (ou solution réalisable) : toute donnée qui satisfait le type de réponse recherché.
- Solution optimale : toute solution dont le coût satisfait le but.

L’exemple du voyageur de commerce (1)

- Le voyageur de commerce (variante « détaillée »)
  - Entrée : ensemble de villes, les distances deux à deux entre villes
  - Type de réponse : un parcours qui visite chaque ville exactement 1 fois et revient au point de départ
  - Coût d’une réponse : la longueur totale du parcours
  - But : minimiser

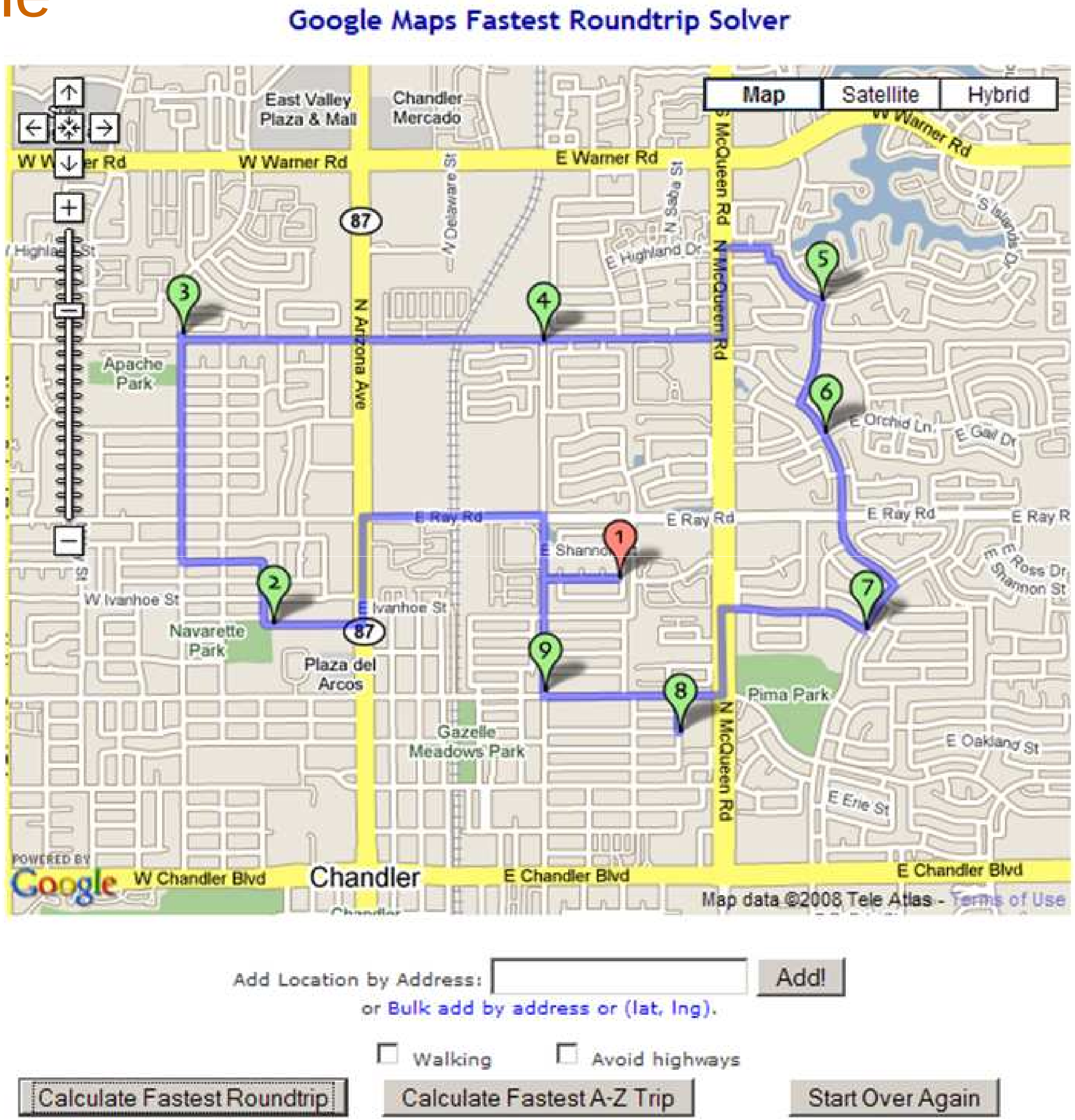
Un exemple


San Marino
Sofiya
Skopje
Roma
Tiranë
Athinai
Tunis
Valletta
Tarabulus

L’exemple du voyageur de commerce (2)

- Le voyageur de commerce (variante « raccourcie»)
  - Entrée : ensemble de villes, les distances deux à deux entre villes
  - Question : quelle est la plus courte distance parcourue par un voyageur de commerce qui doit traverser chaque ville exactement une fois et revenir au point de départ ?
- **Remarques :**
  - On peut demander seulement le coût d’une solution optimale
  - … ou alors la solution elle-même

Un autre exemple



Sommaire

- Types de problèmes
- Résolution et algorithmes gloutons
- Approche intuitive des algorithmes gloutons
  - Principe
  - 1ère application : le problème du Sac à dos
- Variante standard et variations
  - 2ème application : le problème de l'affectation de ressources
  - Correction et preuve

Résolution

- Un algorithme de résolution doit :
  - Construire une ou plusieurs solutions
  - En évaluer le coût
  - Identifier une solution dont le coût s'approche autant que possible du coût optimal

- Du coup :
  - Algorithme exact : qui trouve toujours le coût optimal
  - Algorithme d'approximation : qui n'est pas exact mais garantit un éloignement borné entre le coût fourni et le coût optimal.
  - Heuristique : qui n'offre aucune garantie théoriquement prouvée sur l'éloignement entre le coût fourni et le coût optimal.

Algorithmes gloutons – Pour quels problèmes ?

- S'appliquent sur des problèmes
  - D'optimisation (en général, mais pas toujours)
  - Dont la solution est un ensemble d'éléments, éventuellement muni d'un ordre sur les éléments
  - Dont le coût est la somme des coûts unitaires sur les éléments

- Un exemple : Le voyageur de commerce
  - C'est un problème d'optimisation
  - Une solution est une suite de villes, dans l'ordre de leur visite
  - Son coût est la somme des coûts (distances) pour aller d'une ville déjà visitée à la suivante.

Exemple : le problème du Sac à dos

- Entrée : n objets 1, 2, 3, . . . , n de poids $p_1, p_2, p_3, \ldots, p_n$
- Question : quel est le poids maximum que l'on peut mettre dans le sac à dos en choisissant parmi les objets donnés, sans dépasser la capacité P du sac ?

- Solution : tout ensemble $i_1, i_2, \ldots, i_k$ d'objets parmi 1, 2, …, n t.q.

$$p_{i1}+p_{i2}+\ldots+p_{ik} \leq P$$

- Solution optimale: la solution $i'_1, i'_2, \ldots, i'_{k'}$ qui réalise le poids total maximum, c.à.d.

$$p_{i'1}+p_{i'2}+\ldots+p_{i'k} \geq p_{i1}+p_{i2}+\ldots+p_{ik},$$

pour toute solution $i_1, i_2, \ldots, i_k$

Sommaire

- Types de problèmes
- Résolution et algorithmes gloutons
- Approche intuitive des algorithmes gloutons
  - Principe
  - 1ère application : le problème du Sac à dos
- Variante standard et variations
  - 2ème application : le problème de l’affectation de ressources
  - Correction et preuve

Algorithmes gloutons – Comment ça marche ?

- Principe :
  - Commencer avec une solution partielle vide
  - Chaque étape ajoute un nouvel élément à la solution partielle déjà construite
  - Cet élément est choisi pour être localement le plus prometteur
  - A la fin, l'ensemble obtenu doit être une solution

Sur le Sac à dos (1)

- Principe :
  - Commencer avec une solution partielle vide
  - Chaque étape ajoute un nouvel élément à la solution partielle déjà construite
  - Cet élément est choisi pour être localement le plus prometteur
  - A la fin, l'ensemble obtenu doit être une solution

- Algorithme glouton :

L ← Φ ; poids ←0 ;

O← {1, 2, …, n} ;

tant que (O≠Φ) **faire**

i ← l'objet de O de poids maximum ;

si poids+p_i ≤ P **alors**

L←L∪{i}; Poids←Poids+p_i

finsi

O ← O-{i}

fintantque

Sur le Sac à dos (2)

• Algorithme glouton :	Complexité
L ← Φ ; Poids ←0 ;	O(1)
O← {1, 2, …, n} ;	O(1)
tant que (O≠Φ) **faire**	Répétée n fois
i ← l'objet de O de poids maximum ;	O(n)
si poids+p_i ≤ P **alors**	O(1)
L←LU{i}; Poids←Poids+p_i	
finsi ;	
O ← O-{i}	O(1)
fintantque	Total : O(n^2)

Sur le Sac à dos (3)

- Algorithme glouton :

  $L \leftarrow \Phi$; Poids $\leftarrow 0$;

  $O \leftarrow \{1, 2, \ldots, n\}$;

  tant que $(O \neq \Phi)$ **faire**

  i $\leftarrow$ l'objet de O de poids

  maximum ;

  si (Poids+$p_i \leq P$) **alors**

  $L \leftarrow L \cup \{i\}$; Poids $\leftarrow$ Poids+p_i;

  finsi ;

  $O \leftarrow O-\{i\}$

  fintantque

- Algorithme glouton amélioré

  $O \leftarrow \{1, 2, \ldots, n\}$

  Ordonner les objets de O par ordre décroissant de leur poids

  $L \leftarrow \Phi$; Poids $\leftarrow 0$; $i \leftarrow 1$;

  tant que $(i \leq n)$ **faire**

  si (Poids+$p_i \leq P$) **alors**

  $L \leftarrow L \cup \{i\}$; Poids $\leftarrow$ Poids+p_i;

  finsi;

  $i \leftarrow i+1$;

  fintantque

Sur le Sac à dos (4)

Algorithme glouton amélioré	Complexité
O← {1, 2, …, n}	O(1)
Ordonner les objets de O par ordre décroissant de leur poids	O(n log n)
L ← Φ ; Poids ←0 ; i ← 1;	O(1)
tant que (i≤ n) **faire**	Répétée n fois
si (Poids+p_i ≤ P) **alors**	O(1)
L←LU{i}; Poids←Poids+p_i;	
finsi;	
i ← i+1;	O(1)
fintantque	Total : O(n log n)

Sommaire

- Types de problèmes
- Résolution et algorithmes gloutons
- Approche intuitive des algorithmes gloutons
  - Principe
  - 1ère application : le problème du Sac à dos
- Variante standard et variations
  - 2ème application : le problème de l'affectation de ressources
  - Correction et preuve

Algorithmes gloutons – variante standard

E ensemble de n éléments, $F \subseteq E$ la solution à construire

1. Classer les éléments de E par ordre d'intérêt décroissant

   $$e_1, e_2, \dots, e_n$$

2. $F \leftarrow \Phi$;
3. Pour i de 1 jusqu'à n faire

   si $F \cup \{e_i\}$ est une solution partielle* alors

   $$F \leftarrow F \cup \{e_i\}$$

   finpour

* Cela signifie que l'ensemble ne satisfait pas forcément toutes les propriétés d'une solution, mais permet d'être complété par ajout d'éléments pour aboutir à une solution

Algorithmes gloutons – autres variantes

- Parfois, l'intérêt que présente chaque élément ne peut pas être calculé à l'avance

→ besoin de chercher l'élément le plus prometteur à l'intérieur de la boucle (comme dans l'algorithme glouton initial pour le voyageur de commerce)

- Parfois, le nombre d'éléments de F (ou un autre critère d'arrêt) peut permettre un arrêt de la boucle avant d'avoir vu tous les éléments e_i

→ le « pour » se transforme en « tant que »

- Parfois, le test de solution partielle nécessite l'introduction d'autres variables, tests intermédiaires etc.

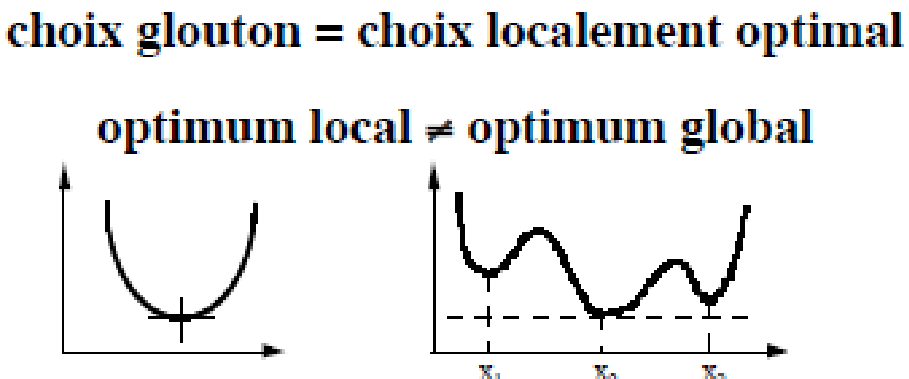
→ peu d'algorithmes gloutons ont, à la fin, la forme standard.

Algorithmes gloutons – Qualité de la solution

- Un algorithme glouton ne fournit pas toujours la solution exacte

- Le voyageur de commerce : pas exact
- Le sac à dos : pas exact

- Pourquoi ?



- Parfois, on a un algorithme d'approximation ♥
- Parfois, on a un algorithme exact ♥♥

Remarque. Le critère d'intérêt est très important.

Un exemple détaillé

- Affectation de ressources

Un loueur de voitures débutant possède une voiture et, sur une semaine, il a noté plusieurs demandes de réservations

$E = \{ e \mid e = [d, f],\ d \leq f\}$, où d (entier) est le jour de début de e

f (entier) est le jour de fin de e

Quelles demandes de réservation le loueur doit-il accepter pour maximiser le nombre de demandes satisfaites ?

Formalisation :

Entrée : $E = \{ e \mid e = [d, f],\ d \leq f\}$ de cardinalité n

Sortie : Ensemble $F \subseteq E$ t.q. toutes les demandes de F soient disjointes et $|F|$ soit maximum. (donc ici coût(F)=|F|).

Un premier algorithme glouton

Affectation de ressources

Entrée : $E = \{ e \mid e = [d, f],\ d \leq f\}$ de cardinalité n

Sortie : Ensemble $F \subseteq E$ t.q. toutes les demandes de F soient disjointes et $|F|$ soit maximum.

1. Classer les éléments de E par dates de début croissantes

   $$e_1, e_2, \dots, e_n \text{ t.q. } d_1 \leq d_2 \leq \dots \leq d_n$$

2. $F \leftarrow \Phi$;
3. Pour i de 1 jusqu'à n faire

   si $F \cup \{e_i\}$ ne contient que des intervalles disjoints alors

   $F \leftarrow F \cup \{e_i\}$ finsi

   finpour

Correction

- Fonctionne parfois

Exemple ?

- … mais pas toujours

Exemple ?

- … et il peut être aussi mauvais (par la différence entre l’optimum et la valeur qu’il retourne) qu’on veut.

Exemple ?

Un deuxième algorithme glouton

- Affectation de ressources

Entrée : $E = \{ e \mid e = [d, f],\ d \leq f\}$ de cardinalité n

Sortie : Ensemble $F \subseteq E$ t.q. toutes les demandes de F soient disjointes et $|F|$ soit maximum.

1. Classer les éléments de E par dates de fin croissantes

$$e_1, e_2, \dots, e_n \text{ t.q. } f_1 \leq f_2 \leq \dots \leq f_n$$

2. $F \leftarrow \Phi$;
3. Pour i de 1 jusqu'à n faire

si $F \cup \{e_i\}$ ne contient que des intervalles disjoints alors

$F \leftarrow F \cup \{e_i\}$ finsi

finpour

Correction

- Fonctionne sur tous les exemples qu'on peut trouver

Exemples ?

- Est-ce qu'il fonctionne toujours ?

Oui.

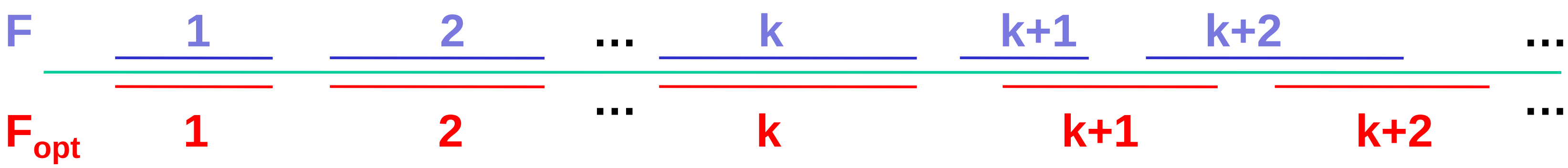
Preuve de la correction (par contradiction).

Principes de preuve (souvent utilisés pour les algorithmes gloutons) :

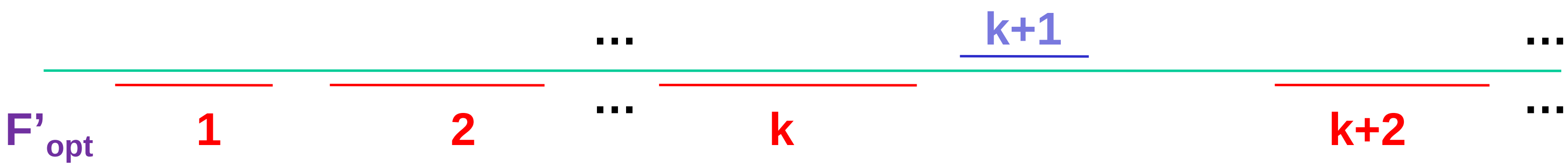
- Soit F la solution fournie par l'algorithme
- Soit F_{opt} une solution optimale aussi proche de F que possible, selon un critère portant sur les éléments de F et F_{opt}.
- Montrer que, si F n'est pas optimale à son tour, alors F_{opt} n'a pas été choisie aussi proche de F que possible (contradiction).

Correction du deuxième algorithme glouton

- Choisir F_{opt} une sol. optimale qui a un maximum d'éléments en commun avec F, càd $|F_{opt} \cap F| \geq |F'_{opt} \cap F|$, $\forall$ F'_{opt} une sol. optimale
- Ordonner les éléments de F_{opt} par ordre croissant de leurs dates de fin (comme c'est déjà le cas pour F, par construction)
- Soit k maximum t.q. les k premiers éléments de F et F_{opt} sont identiques



- Alors F'_{opt} ci-dessous est une sol. optimale t.q. $|F_{opt} \cap F| < |F'_{opt} \cap F|$.



- Contradiction (et fin de la preuve).

Efficacité

1. Classer les éléments de E par dates de fin croissantes

$e_1, e_2, \ldots, e_n$ t.q. $f_1 \leq f_2 \leq \ldots \leq f_n$

2. $F \leftarrow \Phi$;

3. Pour i de 1 jusqu'à n faire

   si FU {e_i} ne contient que des intervalles disjoints

   alors

   $F \leftarrow FU\{e_i\}$

   finsi

   finpour

Complexité :

O(n log n)

O(1)

Répétée n fois

? Test intersection 2 à 2 de tous les intervalles de FU {e_i} ? $O(n^2)$

? Test intersection e_i avec tous les autres intervalles de F ? $O(n)$

? Test intersection ei avec le dernier arrivé dans F (avant e_i) ? $O(1)$

Total : $O(n^3)$ ou $O(n^2)$ ou $O(n \log n)$

L'algorithme glouton – version finale

1. Classer les éléments de E par dates de fin croissantes

   $e_1, e_2, \ldots , e_n$ t.q. $f_1 \leq f_2 \leq \ldots \leq f_n$

2. $F \leftarrow \Phi$; $f \leftarrow 0$ // jour de fin de la dernière demande ajoutée à F

3. Pour i de 1 jusqu'à n faire

   si $d(e_i) > f$ alors // $e_i=[d(e_i),f(e_i)]$

   $F \leftarrow F \cup \{e_i\}$; $f \leftarrow f(e_i)$

   finsi

   finpour

Complexité : O (n log n)

Conclusion

- Les algorithmes gloutons peuvent être très utiles comme première approche.
- … mais ne jamais supposer par défaut que la solution sera exacte.
- Si on peut montrer l'exactitude : c'est très bien, mais c'est rare.
- Sinon, tenter de trouver les failles de l'algorithme glouton pour les corriger et obtenir un autre algorithme, meilleur.