RCP104
METHODES HEURISTIQUES : ALGORITHMES GLOUTONS

PLAN

- **Algorithmes gloutons**
- **Arbre couvrant minimal**
- **Choix d'activités**

1

un problème → plusieurs solutions
une solution → une valeur

problème d'optimisation:
recherche d'une solution de valeur optimale (min ou max)

algorithme de résolution:
- **reconnaissance d'une solution**
- **évaluation d'une solution**
- **sélection d'une des meilleures solutions**

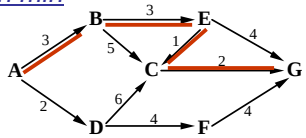
2

problèmes d'optimisation

→ faciles
→ difficiles

EXEMPLES

E1) chemin min



problème facile
(v=9;ABECG)

3

E2) sac-à-dos (en nombres entiers)

problème "assez" difficile

aliments	A	B	C
poids unitaire hg	6	4	3
valeur nutritive unitaire	14	10	6
poids max des aliments: 17			

2 solutions optimales:

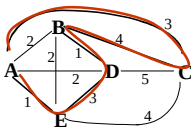
1.A + 2.B + 1.C → v = 40 (p=17)

4.B → v = 40 (p=16)

4

E3) voyageur de commerce

problème difficile



ici, 2 solutions optimales

ACBDEA et ACEBDA v = 12

nombre de solutions: 24 (n-1!)

5

problème de petite taille
→ énumération possible

problème de grande taille
→ énumération impossible

optimisation discrète ≠ optimisation continue

6

problèmes faciles →
algorithme de complexité polynomiale

problèmes "difficiles" →

- **tous les algorithmes connus sont de complexité exponentielle**
- **problèmes dits NP-complets ou NP-difficiles**

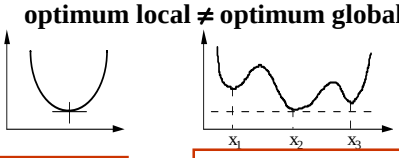
7

ALGORITHMES GLOUTONS

8

choix glouton = choix localement optimal

optimum local ≠ optimum global



fonction concave (ou convexe)
optimum local = optimum global
continu ≠ entier

x_1 et x_3 : optima locaux

x_2 optimum global (et local)

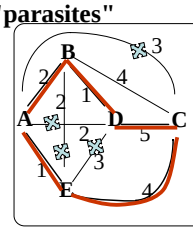
9

algorithme glouton: pas toujours optimal

EXEMPLE

un algo glouton pour le PVC:

- **trier les arêtes (coûts croissants)**
- **sélectionner dans l'ordre les arêtes non "parasites"**



([A,E],[B,D],[A,B],
~~[A,D]~~,[B,E],~~[A,C]~~,
~~[E,D]~~,[C,E],[C,D])

[A,B,D,C,E,A]
valeur 13

(opt=12 [ACEBDA])

10

algorithme glouton: à chaque étape
choix le plus intéressant <u>à cet instant</u>

→ **facile à concevoir**

→ **difficile de vérifier l'optimalité**

→ **efficace (faible complexité)**

11

ARBRE COUVRANT MINIMAL

12

LE PROBLÈME

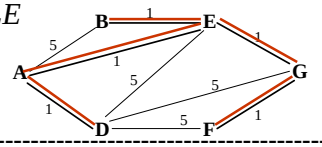
relier des objets avec une longueur totale minimale des liens

graphe valué associé:

- **sommets = objets**
- **arête = lien possible**
- **poids d'une arête = longueur du lien**

13

EXEMPLE



solution optimale:

sans cycle et connexe → ARBRE

par tous les sommets → COUVRANT

de longueur totale min → MINIMAL

14

6.3.2 ALGORITHME DE KRUSKAL

rappel: arbre → m = n-1 arêtes

entrée: G = (X,U,P)
n sommets, m arêtes

sortie: A arbre couvrant min de G

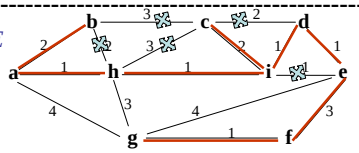
15

---

procedure **kruskal** (in **G**: **graphe;** out **A: arbre):**
entier k=0; **{arêtes} V=∅**;
début
trier les arêtes de G par ordre de poids croissant ;
tant que **k < n-1** faire
  parcourir la liste triée ;
  sélectionner la première arête, w, qui ne forme pas de cycle avec les précédentes ;
  k = k+1 ;
  V = V ∪ {w} ;
fait ;
A = graphe (X,V) ;
fin

16

EXEMPLE



liste triée:

ah	de	di	ei	hi	fg	ab	bh	ci	cd	bc	ch	ef	gh	ag	ge
1	1	1	1	1	1	2	2	2	2	3	3	3	3	4	4
x	x	x	-	x	x	x	-	x	-	-	-	x	STOP		

n = 9 → stop après 8 sélections

P(A) = (1+1+1+1+1+2+2+3) = 12

17

- **implémentation peu facile**
- **complexité O(m logm)** (TRI)

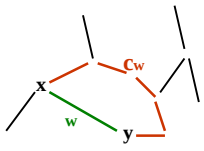
18

6.3.3 PREUVE DE L 'OPTIMALITE

G=(X,U) le graphe
A=(X,V) l'arbre couvrant obtenu par Kruskal
p(u) : poids de l'arête u

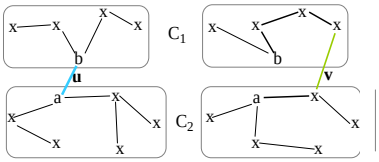
propriété 1

soit w ∈ U -V, w=[x,y] et
c_w : chaîne de x à y dans A;
alors, $p(w) \geq \max_{u \in c_w} p(u)$



Car s'il existait une arête a de c_w de poids plus fort que p(w), w aurait été sélectionnée avant a (w ne pouvant former de cycles avec les arêtes précédemment choisies)

A': arbre optimal de poids p(A')
montrons que p(A) = p(A')
soit u ∈ A'-A



	A'=(X,V')	A=(X,V)
u ∈ V'-V	relie C_1 et C_2 dans A'	
v ∈ c_u		relie C_1 et C_2 dans A

propriété 1 ⇒ p(u) ≥ p(v) et
A' minimal ⇒ p(v) ≥ p(u)
⇒ p(u) = p(v)

soit A"=A'+{v}-{u}: p(A")=p(A')

→ A"optimal et v ∈ A"

soit u'∈ A"-A,... (idem... → A''')
...
fin: A'''''' = A et p(A'''''') = p(A') = p(A)

Fin preuve

CHOIX D 'ACTIVITES

6.4.1 LE PROBLÈME

A = {a_1,a_2,..,a_n} activités
a_i ∈ [d_i,f_i[d_i début, f_i fin de a_i
a_i et a_j compatibles si $d_i \geq f_j$ ou $d_j \geq f_i$

problème:
choisir le plus grand nombre d'activités compatibles

6.4.2 ALGORITHME CHOIX_ACTIVITÉ

choix glouton : maximiser le temps restant après l'activité choisie

Algorithme **choix_activité** (in **A: liste initiale d'activités**;
out **A*: liste d'activités choisies):**

début

trier et numéroter les a_i dans l'ordre croissant des f_i ; $f_1 \leq f_2 \leq \ldots \leq f_n$

A* = {a_1} ;

j = 1; *j représente la dernière activité sélectionnée*

pour **i = 1** à **n** faire

si **$d_i \geq f_j$** alors

A* = A*∪{a_i} ;

j = i ;

finsi;

fait ;

fin

25

complexité: **O(nlogn)+O(n)**

tri *boucle*

⇒ O(nlogn)

EXEMPLE

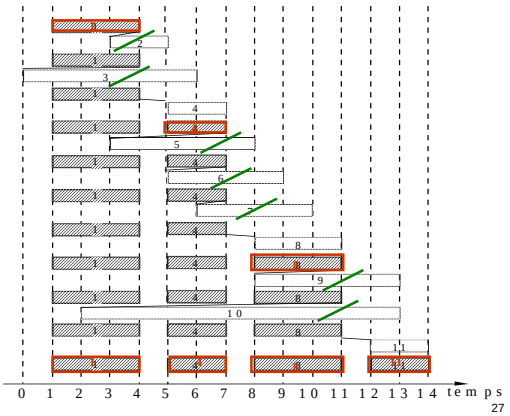
d_i date de début de l'activité a_i

f_i date de fin de l'activité a_i

activités triées selon les f_i

i	1	2	3	4	5	6	7	8	9	10	11
di	1	3	0	5	3	5	6	8	8	2	12
fi	4	5	6	7	8	9	10	11	12	13	14

26



27

OPTIMALITE

Soit B solution optimale B ≠ A* card B ≥ card A*

B ordonné selon les f_i:

$a_{k1}, a_{k2}, .., a_{kn}$ avec $f_{k1} \leq f_{k2} \leq \ldots \leq f_{kn}$

si $a_{k1} \neq a_1$, soit **B'= B-{a_{k1}}+{a_1}**

- $f_1 \leq f_{k1} \leq f_{k2} \leq \ldots \leq f_{kn}$ → B' admissible

- card B' = card B ⇒ B' optimal

sinon B'= B

Conclusion : il existe une solution optimale commençant par un choix glouton

28

problème restant = problème initial avec
A' = {$a_i \in$ A t.q. $d_i \geq f_1$}

→ choix glouton suivant de B': a_2

...

fin: B'''' = A*

et card B'''' = card B = card A*

Fin preuve

29