Algorithmes gloutons

1. Optimisation combinatoire
2. Allocation de ressources
3. Arbre recouvrant minimum
4. Caractérisation des situations où glouton donne l'optimum
5. Matroïdes

Optimisation Combinatoire: problème abstrait

Soit E un ensemble fini

Une valuation v des éléments de E, $v : E \to \mathcal{R}^+$

Une famille $\mathcal{F}$ de parties de E, $\mathcal{F} \subseteq E$

Problème Trouver $F \in \mathcal{F}$ tel que

$$\sum_{f \in F} v(f)$$

soit maximal.

Remarques

1. Le nombre d'éléments de $\mathcal{F}$ est une fonction exponentielle de n.
2. Vérifier si $F \in \mathcal{F}$ a coût de l'ordre de $|F|$ (nombre d'éléments de F), ou bien est polynomial en $|F|$.
3. Le plus souvent la famille $\mathcal{F}$ est close par passage aux sous-ensembles :

$$F \in \mathcal{F} \text{ et } G \subset F \quad \Rightarrow \quad G \in \mathcal{F}$$

Exemples

1. **:"subset-sum" :** On a n objets de poids $p_1, p_2, \ldots, p_n$ et on veut en mettre dans un sac de poids total inférieur ou égal à P. Il s'agit de trouver le poids maximum que l'on peut mettre dans le sac.
$$v(i) = p_i, \quad \mathcal{F} \quad = \{F \mid \sum_{i \in F} p_i \ \leq \ P\}.$$
2. Formation d'une liste de candidats : On se propose de constituer une liste de personnes aussi large que possible à prendre parmi $x_1, x_2, \ldots, x_n$, en tenant compte d'incompabilités correspondant à des paires qui refusent energiquement de se retrouver ensemble sur la liste.
$$v(x) = 1, \quad \mathcal{F} \quad = \{F \mid \forall x, y \in F, \ \{x, y\} \notin U\}$$

Exemples (suite).

- "Bin-packing" : On doit placer des objets de tailles $t_1, t_2, \ldots t_n$ dans des boîtes de capacité limitée. Il s'agit de déterminer le nombre minimum de boîtes qui peuvent contenir tous les objets.
- Fiabilité de réseau : On a un certain nombre de points à relier par un réseau ; chaque liaison entre deux points a une certaine fiabilité. Il s'agit de trouver l'ensemble des liens à retenir de façon telle que tous les points soient reliés entre eux, qu'il n'y ait pas de circuit et que la somme des fiabilités soit maximale.
- Ordonnancement : On a des tâches à réaliser sur des machines et il s'agit de déterminer pour chaque machine les tâches qui doivent y être accomplies et dans quel ordre de façon à maximiser une certaine fonction.

L'algorithme glouton général

1. Classer les éléments de E par valuations décroissantes :

$$v(e_1) \geq v(e_2) \ldots \geq v(e_n)$$

2. Initialisation : Poser $F \leftarrow \emptyset$

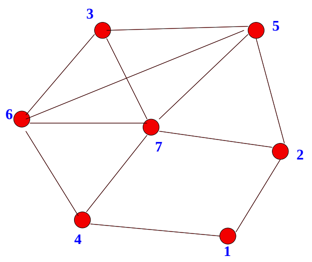
3. Pour i allant de 1 à n faire Si $F \cup \{e_i\} \in \mathcal{F}$ Alors $F \leftarrow F \cup \{e_i\}$

L'algorithme glouton ne marche pas toujours ..

Liste sans incompatibilité

Une idée est de classer les personnes par ordre de nombre d'incompatibilités croissant

On n'obtient que deux personnes $\{1, 3\}$, alors que trois est possible: $\{2, 3, 4\}$.



Réservation de salle : le problème

- n demandes $e_1, e_2, \ldots e_n$ de réservation de salle
- demande e_i :

  date de début d_i, date de fin f_i
- But : sous-ensemble F de cardinal maximum et qui satisfait :

$e_i, e_j \in F \Rightarrow \;]d_i, f_i[$ et $]d_j, f_j[$ disjoints

Réservation de salle : algorithme

- Classer les e_i par dates de fins croissantes
- Initialisation : $F \leftarrow \{e_1\}$
- Pour i allant de 2 à n faire
- Si e_i n'intersecte pas le dernier élément de F Alors
- $F \leftarrow F \cup \{e_i\}$

Réservation de salle : analyse

- Soient: F donné par l'algorithme glouton, G optimal tel que $|F \cap G|$ soit maximum
- Par l'absurde: si $G \neq F$, soit g premier élément g de $F \setminus G$.

Graphes: définitions

Définition Graphe $G = (V, E)$: V sommets, E d'arêtes = paires de sommets.

chemin : suite d'arêtes

$$e_1 = \{x_1, x_2\}, e_2 = \{x_2, x_3\}, \ldots e_p = \{x_p, x_{p+1}\}$$

cycle: chemin tel que $x_1 = x_{p+1}$

chemin *élémentaire* : ne passe pas deux fois par le même sommet

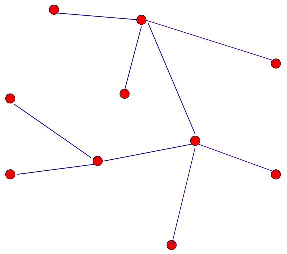
Graphe connexe : toute paire de sommets est reliée par un chemin.

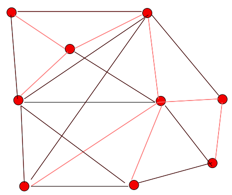
Arbre : graphe connexe sans cycle.

Arbres

Les propriétés suivantes sont équivalentes pour un graphe G ayant n sommets :

1. G est un arbre
2. G est connexe minimal
3. G est sans cycle maximal
4. G est connexe et a $n-1$ arêtes
5. G est sans cycle et a $n-1$ arêtes
6. Toute paire de sommets de G est reliée par un unique chemin





Arbres couvrants

Notations

- $G = (V, E)$, $|V| = n$, $|E| = m$
- (V, T) arbre couvrant de G

Propriétés

1. $|T| = n - 1$
2. Chaque arête e qui n'est pas dans T crée un cycle C_e unique avec T.

Arbre couvrant de poids maximum

G graphe où chaque arête e a un poids $p(e)$. But: arbre couvrant de poids total maximum

Problème d'optimisation combinatoire avec :

$\mathcal{F}$ est la famille des partie de E qui ne contiennent pas de cycle

Algorithme de Kruskal

1. Classer les arêtes par poids décroissants : $p(e_1) \geq p(e_2) \ldots \geq p(e_n)$

2. Initialisation: $T \leftarrow \emptyset$

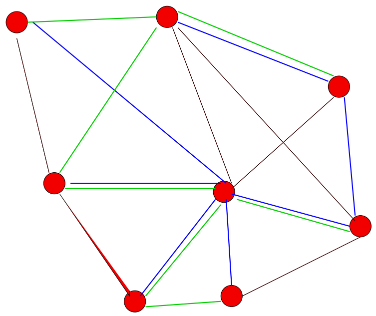
3. Pour i allant de 1 à n faire Si $T \cup \{e_i\}$ n'a pas de cycle Alors $T \leftarrow T \cup \{e_i\}$

Preuve de validité de Kruskal

Lemme

Soient T et U deux arbres recouvrants de G, soit $a \in U$, $a \notin T$ alors il existe $b \in T$ tel que $U - a + b$ soit un arbre recouvrant.

De plus b peut être choisi dans le cycle formé par des arêtes de T et a.



Algorithme de Kruskal : analyse

- Soient T l'arbre obtenu par l'algo glouton U un arbre de poids maximal tel que $|T \cap U|$ soit maximum
- Par l'absurde. Si $T \neq U$: soit $e \in T \setminus U$...

Algorithme glouton général : analyse

L'algorithme glouton donne une solution optimale quelle que soit la fonction v si et seulement si la famille $\mathcal{F}$ vérifie la propriété suivante :

Si F et G sont deux ensembles de $\mathcal{F}$ tels que G a plus d'éléments que F, alors il existe un élément x de $G \setminus F$ tel que $F \cup \{x\} \in \mathcal{F}$.

$$\Longrightarrow$$

- Soient F et G donnés, $|G| > |F|$. On définit v par:
- On vérifie: $v(G) \geq v(F)$. Glouton $\Longrightarrow F' \supseteq F$, $F' \neq F$.

$\Longleftarrow$

- Soit F obtenu par l'algo glouton, G appartenant à $\mathcal{F}$, optimal et tel que $|F \cap G|$ maximum ; si $F \neq G$, soit $e \in G \setminus F$, e premier élément (par ordre de v décroissant)...

Matroïdes : définitions

La famille $\mathcal{F}$ de sous-ensembles de E forme un matroïde si les trois conditions suivantes sont satisfaites :

1. La famille $\mathcal{F}$ n'est pas vide
2. Si $F \in \mathcal{F}$ et $G \subset F$ alors $G \in \mathcal{F}$
3. Si $F, G \in \mathcal{F}$ et $|G| > |F|$ alors $\exists x \in G \setminus F$ tel que $F \cup \{x\} \in \mathcal{F}$

Les éléments de $\mathcal{F}$ sont appelés les *ensembles indépendants* du matroïde.

Exemples de matroïdes

- Ensemble de vecteurs libres d'un espace vectoriel
- Arcs sans origine commune dans un graphe orienté
- Ensemble de représentants distincts
- Ensemble d'arêtes dans un graphe biparti

Programmation dynamique

1. **Plus courts chemins à partir d'un sommet**
2. **Plus courts chemins entre tous les sommets**
3. **Nombres de Fibonacci**
4. **Sac à dos**
5. **Alignement de séquences**
6. **Programmation dynamique**

Plus court chemin à partir d'un sommet

Entrée : graphe orienté aux arcs valués et un sommet x_0
Sortie : pour chaque sommet x : la longueur du plus court chemin de x_0 à x, et la suite des sommets constituant ce plus court chemin

Structure de données :

- `d[x]` longueur du chemin de x_0 à x
- `pere[x]` sommet qui précède x sur le chemin de x_0 à x

Calcul du chemin à partir de la structure de données :

chemin(x_0, x)=chemin(x_0,père(x))@ (père(x),x)

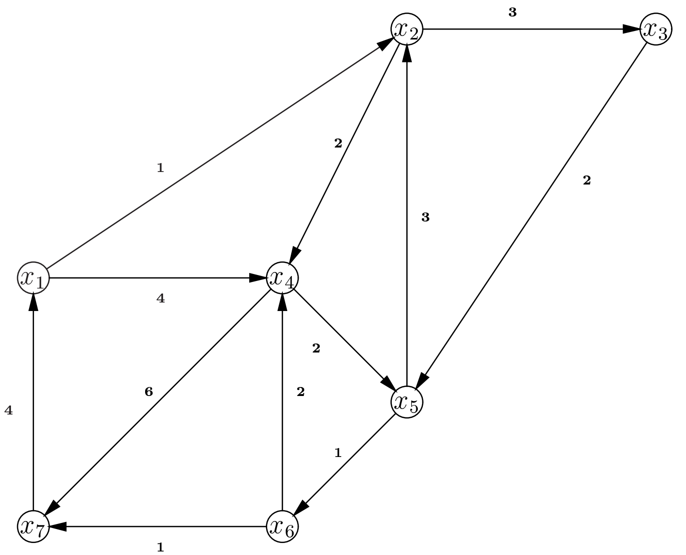


Figure 1: Calcul de plus courts chemins

Algorithme de Dijkstra

Initialisation :
$A \leftarrow \emptyset$
$d[x_0] \leftarrow 0$
$d[x] \leftarrow \infty$ **pour** $x \neq x_0$

Tant que $A \neq V$ **:**
  Soit y **l'élément de** $V \setminus A$ **avec** $d[y]$ **minimum**
  $A \leftarrow A \cup \{y\}$
  Pour tout z **voisin de** y**, relaxer l'arc** (y, z)**.**

Relaxer l'arc (y, z)

Si $d[y] > d[x] + \ell(x, y)$
Alors
  $d[y] \leftarrow d[x] + \ell(x, y)$
  père$[y] \leftarrow x$

Algorithme de Dijkstra : analyse

Invariant : $d[y]$ **représente la longueur d'un chemin de** x_0 **à** x**.**

Invariant : **Si** $y \in A$**, alors** $d[y]$ **est la longueur du plus court chemin de** x_0 **à** y**, et pour tout voisin** z **de** y**, on a considéré le plus court chemin de** x_0 **à** z **qui se termine par l'arc** (y, z)**.**

Algorithme de Dijkstra : complexité

Structure de données pour $V \setminus A$**: Ensemble avec valeurs** $d[x]$**.**
Opérations:

1. **Trouver et supprimer l'élément plus petite valeur**
2. **diminuer la valeur d'un élément**

Tas : **ces opérations on coût** $O(\log n)$**.**

Complexité : $O(n + m \log n)$**.**

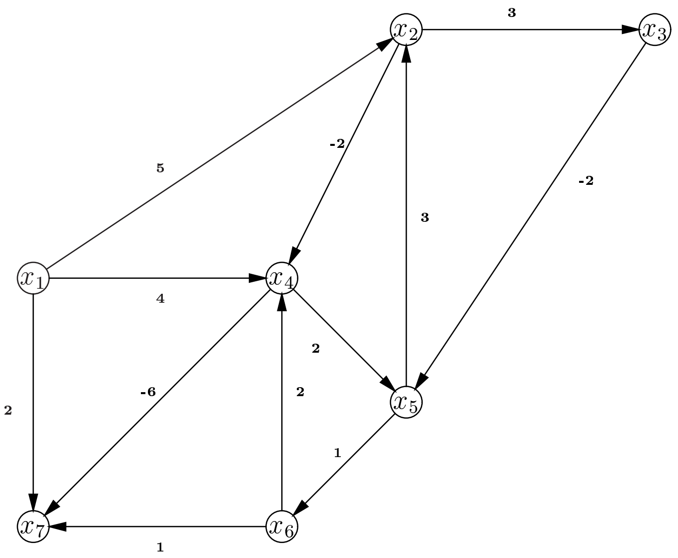


Figure 2: Arcs négatifs

Arcs négatifs : Algorithme de Ford

Pour i **allant de 1 à** $n-1$

  Pour tout arc `(x,y)`

    `Relaxer(x,y)`

Analyse : **Vérifier qu'il n'y a plus de relaxation possible**

Plus courts chemins entre tous les sommets

Soit $\delta_k(x,y)$ la longueur du plus court chemin entre x et y qui soit composé d'au plus k arcs.

But : calculer $\delta_n(x,y)$ pour tout x, y.

Définition récursive :

$$\delta_0(x,y) = \begin{cases} 0 & \textbf{si } x = y \\ \infty & \textbf{sinon} \end{cases}$$

$$\delta_k(x,y) = \min_{z:(x,z)\in E}(\ell(x,z) + \delta_{k-1}(z,y)).$$

Implantation : Tableau $\delta[k,x,y]$.

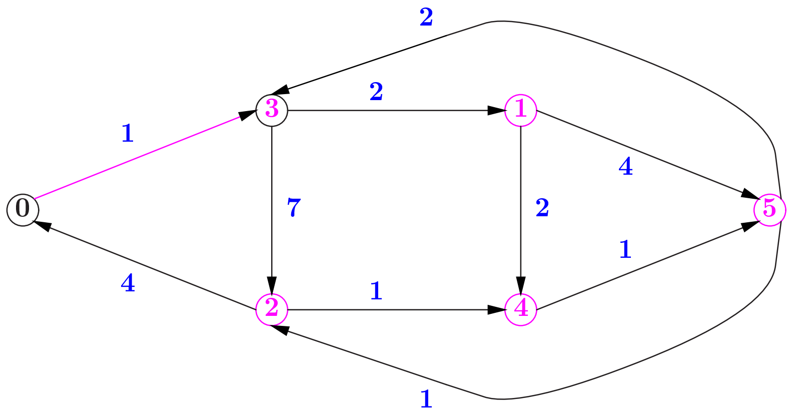


Figure 3: Tous les chemins

Algorithme de Roy-Warshall

Pour k allant de 1 à $n-1$
  Pour i allant de 1 à n
    Pour j allant de 1 à n
      Si $\delta[i,j] > \delta[i,k] + \delta[k,j]$
        Alors $\delta[i,j] \leftarrow \delta[i,k] + \delta[k,i]$

Suite de Fibonacci

Définition récursive : $f_0 = f_1 = 1$, $f_n = f_{n-1} + f_{n-2}$.

Implantation en récursif : complexité exponentielle.

Implantation en itératif : tableau $f[0\ldots n]$, complexité polynomiale.

Sac à dos

Entrée : budget B, et n objets. Objet i: valeur v_i, prix p_i.

But : choisir un sous-ensemble des objets de valeur totale maximale tout en dépensant au plus B.

Définition récursive : Soit $f(i,b)$ la plus grande valeur atteignable par choix dans $\{1,\ldots,i\}$ avec budget b.

$$\begin{aligned} f(0,b) &= 0 \\ f(i,b) &= \max(f(i-1,b), v_i + f(i-1,b-p_i)). \end{aligned}$$

But : $f(n,B)$.

Implantation : tableau $f[0\ldots n, 0\ldots B]$.

Complexité : $O(nB)$.

Exemple

Objets	1	2	3	4	5	6	7	8
Prix	2	3	5	2	4	6	3	1
Valeurs	5	8	14	6	13	17	10	4

Budget 12

Alignements de séquences

Etant donné deux mots sur l'alphabet $\Sigma = \{A, C, G, T\}$ on cherche un alignement qui optimise une fonction qui mesure la similarité

Similarité : $p : (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \to \mathbf{R}_+$

$$p(a,a) > 0, \qquad p(a,b) < 0 \text{ si } b \neq a.$$

Exemple : GATGCAT et ATCGAT

G	A	T	–	G	C	A	T
–	A	T	C	G	–	A	T

Algorithme d'alignement

Entrée : $u_1 \cdots u_n$ et $v_1 \cdots v_m$ mots sur $\{A, C, G, T\}$.

Soit $s(i,j)$ valeur du meilleur alignement entre $u_1 \cdots u_i$ et $v_1 \cdots v_j$.

Définition récursive :

$$\begin{aligned} s(i,0) &= \sum_{k \le i} p(u_k, -) \\ s(0,j) &= \sum_{k \le j} p(-, v_k) \\ s(i,j) &= \max \begin{cases} s(i-1, j-1) + p(u_i, v_j) \\ s(i-1, j) + p(u_i, -) \\ s(i, j-1) + p(-, v_j) \end{cases} \end{aligned}$$

But : $s(n,m)$

Implantation : Tableau $s[0\ldots n, 0\ldots m]$.

Principe d'un algorithme en espace linéaire

- **Le calcul des meilleurs scores peut se faire sur une seule ligne**
- **Le problème est de retrouver l'alignement une fois que l'on a le score**
- **On procède par divide and conquer**
- **Pour aligner** u **et** v **on prend** i **comme la moitié de la longueur de** u
- **On calcule les valeurs** a_j **des meilleurs alignements de** $u_1, u_2, \ldots u_{i-1}$ **avec tous les facteurs gauches de** v
- **On calcule les valeurs** b_j **des meilleurs alignements de** $u_{i+1}, u_{i+2}, \ldots u_n$ **avec tous les facteurs droits de** v
- **On cherche le meilleur** $a_{j-1} + b_{j+1} + p(u_i, v_j)$

Calcul de la complexité de l'algorithme en espace linéaire

- **On montre que le nombre de comparaisons est** $C(m,n) \leq 2mn$
- **Vrai pour** $m = 1$
- $$C(m,n) \leq \frac{mn}{2} + \frac{mn}{2} + C(m/2, j) + C(m/2, n-j)$$

Quand peut-on utiliser la programmation dynamique ?

1. $\exists$ **un ordre partiel sur les problèmes tel que :**
   - **on sait résoudre le problèmes minimaux**
   - **tout problème** P **non minimal se résoud au moyen de solutions de certains problèmes** P'**, avec** $P' < P$
2. **Le calcul pour** P **à partir des solutions pour les** P' **se fait en temps polynomial en** $|P|$**.**
3. **Le nombre total de sous-problèmes,** $|\{P' | P' < P\}|$**, est polynomial en** $|P|$**.**

Flots et Couplages

1. Flots et Coupes
2. Algorithmes de Ford-Fulkerson, Dinic, Edmonds-Karp
3. Applications des flots
4. Couplages
5. Mariages Stables

Définitions

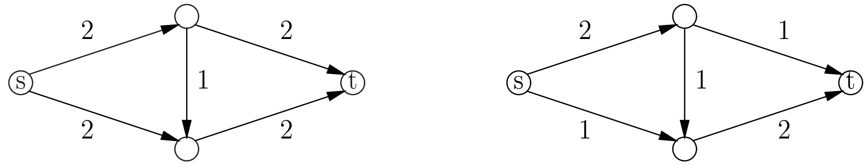
Réseau de transport : Graphe orienté $G = (V, E)$, avec deux sommets particuliers s et t, tels que s n'a pas d'arc entrant et t n'a pas d'arcs sortant. Chaque arc a a une *capacité* $c(a)$.

Un *flot* est une application ϕ de A dans $\mathcal{N}$ telle que :

1. Le flot traversant un arc n'excède pas sa capacité : $\phi(a) \leq c(a)$
2. Pour tout sommet $u \neq s, t$, le flot entrant dans u est égal au flot sortant de u.

La *valeur* du flot est égale à la somme des flots sortants de s.

Problème : trouver un flot de valeur maximale.



Un réseau de transport

Un flot sur le graphe

Figure 4: Flot saturé non maximal

Un arc est *saturé* si $\phi(a) = c(a)$. Un flot est saturé si tout chemin de s à t possède un arc saturé.

Coupes

Une *coupe* est une partition de V en deux sous-ensembles disjoints $V = Y \cup Z$ tels que $s \in Y$ et $t \in Z$. Sa *capacité* est la somme des capacités des arcs qui ont leur origine dans Y et leur extrémité dans Z.

Pour un flot ϕ et une coupe Y, Z on pose

$$\phi(Y, Z) = \sum_{a \in (Y,Z)} \phi(a) \text{ et } \Delta(Y, Z) = \phi(Y, Z) - \phi(Z, Y).$$

Flots et coupes

Pour toute coupe (Y, Z) et tout flot, $\Delta(Y, Z)$ est égal à la valeur du flot.

Preuve : par récurrence sur $|Y|$

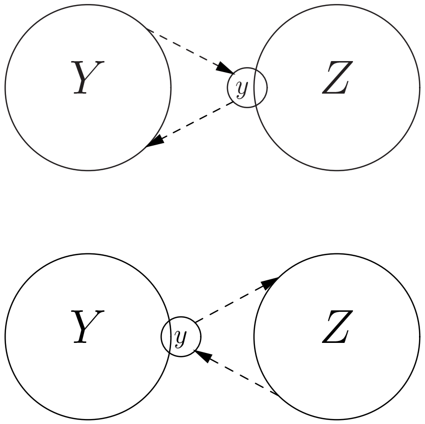


Figure 5: Conservation du flot

Lemme

Pour toute coupe (Y, Z) et tout flot, la valeur du flot est inférieure ou égale à $C(Y, Z)$.

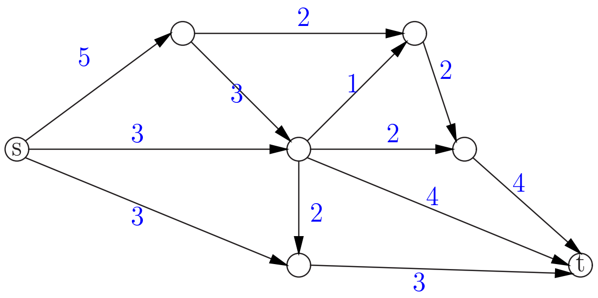
Théorème de Ford-Fulkerson

Le flot maximal a pour valeur la capacité de la coupe minimale.

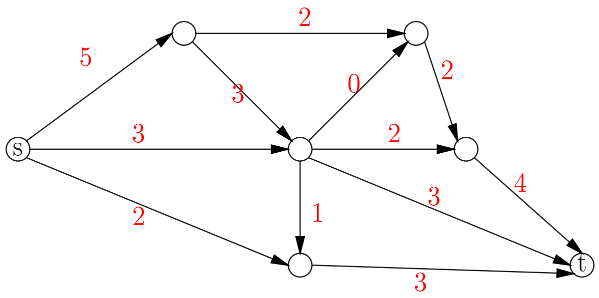
Graphe des augmentations

Etant donné un flot ϕ, on définit un graphe G_ϕ de sommets V et tel que :

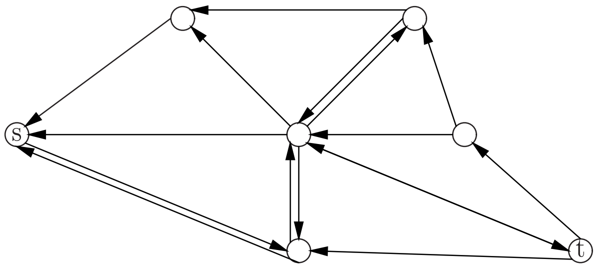
- si $\phi(a) < c(a)$ alors on met l'arc a dans G_ϕ
- si $\phi(a) > 0$ alors on met l'arc inverse de l'arc a dans G_ϕ



Un graphe et ses capacités



Un flot sur ce graphe



Le graphe des augmentations

Lemmes

S'il existe un chemin joignant s à t dans G_ϕ, alors le flot peut être amélioré.

S'il n'existe pas de chemin joignant s à t dans G_ϕ, alors il existe une coupe (Y, Z) de capacité égale à la valeur du flot.

Algorithme de Ford et Fulkerson

Initialisation : On calcule un flot quelconque (par exemple nul)

Boucle : Tant qu'il existe un chemin de s à t dans le graphe des augmentations, on augmente le flot

Sortie : On obtient ainsi le flot maximal

La coupe minimale peut être obtenue en considérant l'ensemble des sommets atteignables à partir de s dans le graphe des augmentations

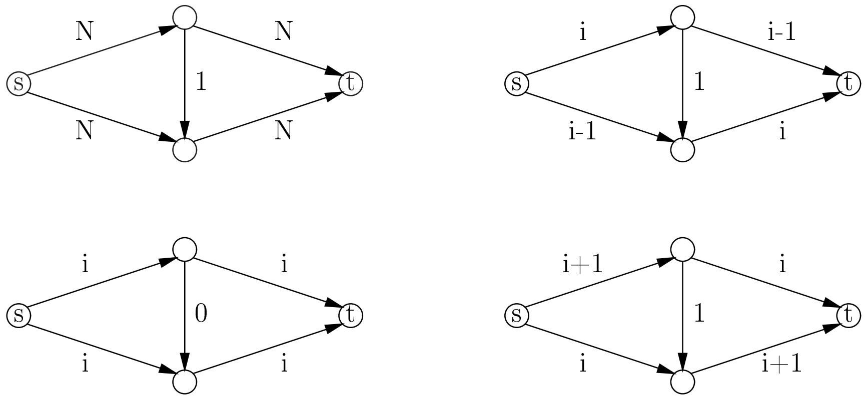


Figure 6: Un exemple de calcul très lent

Algorithme de Dinic, Edmonds et Karp

Initialisation : On calcule un flot saturé

Tant qu'il existe un chemin dans le graphe des augmentations, on augmente le flot en utilisant le *plus court chemin* de s à t (parcours en largeur)

Applications

1. Distribution de denrées
2. Profit maximal pour certains problèmes d'ordonnancement
3. Coloration de cartes

Distribution de denrée

Entrée :

- Les centres de distribution $A_1, A_2, \ldots, A_p$, stockent une quantité $a_1, a_2, \ldots, a_p$ d'une certaine denrée.
- Les clients $B_1, B_2, \ldots, B_q$ demandent une quantité $b_1, b_2, \ldots, b_q$ de la denrée.
- Le client B_i ne peut être livré que par certains des centres de distributions.

Sortie : Déterminer l'approvisionnement maximal de l'ensemble des clients.

Réduction à un problème de flot

sommets $\{s, A_1, \ldots, A_p, B_1, \ldots B_q, t\}$

arc (s, A_i) de capacité a_i

arc (B_j, t) de capacité b_j

arc (A_i, B_j) si le centre de distribution A_i peut livrer de la marchandise à B_j; capacité non bornée.

Problème de choix d'objets à emporter

Entrée :

- On peut emmener dans une navette spatiale un certain nombre d'instruments $A_1, A_2, \ldots, A_p$; coût de transport c_i
- Commandes d'expériences dans l'espace $B_1, B_2, \ldots, B_q$; le gain réalisé si l'on effectue l'expérience B_j est noté b_j
- Pour pouvoir réaliser B_i, il faut d'avoir emporté un sous-ensemble de l'ensemble des instruments

Sortie : Déterminer quels instruments emporter et quelles expériences réaliser pour maximiser le gain

Réduction à un problème de coupe

sommets $\{s, A_1, \ldots A_p, B_1 \ldots B_q, t\}$

arc (s, A_i) de capacité c_i

arc (B_j, t) de capacité b_j

arc (A_i, B_j) si l'expérience B_j nécessite l'instrument A_i ; capacité non bornée

Coloriage de carte

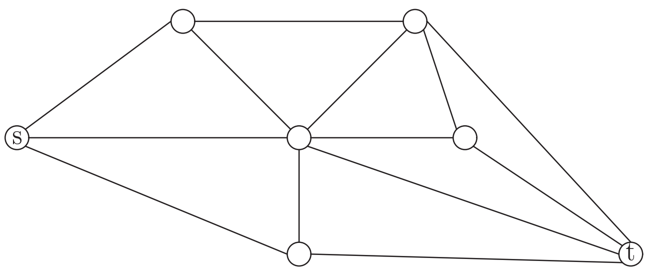
Problème : colorier, avec un nombre minimal de couleurs, avec les pays d'une carte de géographie de façon telle que deux pays voisins aient des couleurs distinctes.

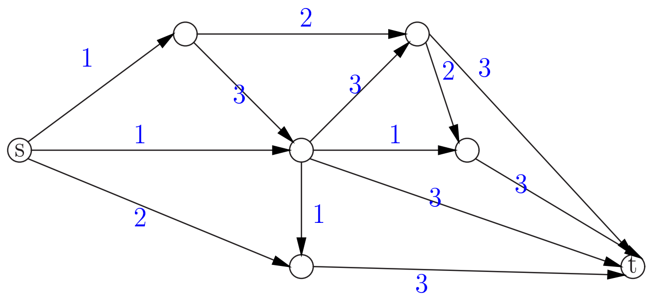
On considère le graphe dont les sommets sont les points situés sur au moins deux frontières

Les arêtes sont déterminées par les frontières

Solution

- Flots dans Z/nZ
- On se donne une orientation des arêtes du graphe
- On attribue des valeurs dans Z/nZ aux arêtes, dont aucune n'est nulle.
- On respecte la condition que pour chaque sommet la somme des valeurs des arcs entrants est égale à la somme des valeurs des arcs sortants (dans Z/nZ)
- La coloration s'obtient à partir du flot en attribuant une couleur quelconque à l'une des faces et en ajoutant la valeur du flot sur un arc à la couleur de la face de l'autre coté de l'arc (ou en retranchant suivant le sens de parcours de l'arc)





Un flot dans $Z/4Z$

Couplage maximum dans un graphe biparti

Un graphe non orienté est *biparti* s'il existe une partition des sommets en deux sous ensembles tels que toute arête ait une extrémité dans chacun des deux sous-ensembles.

Couplage du graphe : ensemble d'arêtes sans extrémités communes.

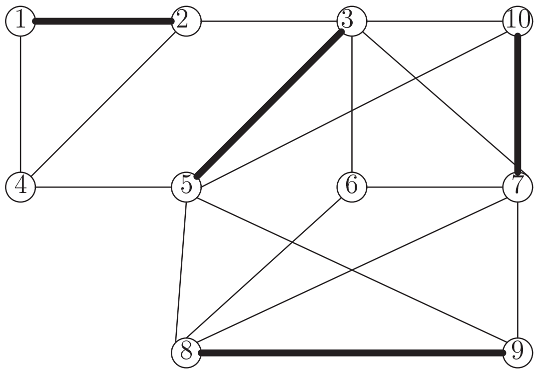


Figure 7: Un couplage

Réduction à un problème de flot

On ajoute deux sommets s et t, on relie s à tous les sommets de Y et tous les sommets de Z à t, on oriente les arêtes de Y vers Z, la capacité est de 1 pour tous les arcs.

Chemins alternants

- Les chemins dans le graphe des augmentations deviennent ici des chemins alternants
- Chemins formés alternativement d'une arête extérieure au couplage et d'une arête appartenant au couplage
- Arête extérieure : on peut augmenter le flot. Arête du couplage : on peut diminuer le flot
- On peut montrer que le couplage est maximum si et seulement si il n'existe pas de chemin alternant joignant deux sommets non saturés

Théorème du couplage maximal

Un couplage est maximal si et seulement s'il n'existe pas de chemin alternant joignant deux sommets non couplés

Généralisation : mariages stables

Entrée : Deux ensembles $X = \{x_1, x_2, \ldots, x_p\}$ et $Y = \{y_1, y_2, \ldots, y_p\}$.

A chaque x_i est associée une liste ordonnée F_i d'éléments de Y.

A chaque y_j est associée une liste ordonnée G_j d'éléments de X.

On travaille dans le graphe biparti où les arêtes sont les couples $\{x, y\}$ tels que y figure dans la liste F_x et x dans la liste G_y

Un couplage C est *stable* si et seulement si pour tout couple x, y qui n'est pas dans C on n'a pas : x préfère y à sa partenaire, et y préfère x à son partenaire.

Mariages stables

Considérons le cas $p = q = 3$ et les listes

$$
\begin{array}{lllll}
F_1 : & y_2 & y_1 & y_3 \\
F_2 : & y_1 & y_3 & y_2 \\
F_3 : & y_1 & y_2 & y_3
\end{array}
\qquad
\begin{array}{llll}
G_1 : & x_1 & x_3 & x_2 \\
G_2 : & x_3 & x_1 & x_2 \\
G_3 : & x_1 & x_3 & x_2
\end{array}
$$

Le couplage $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ n'est pas stable.

Le couplage $(x_1, y_1), (x_2, y_3), (x_3, y_2)$ est stable.

Algorithme

On initialise $\alpha(x)$ au premier élément de la liste F_x et $\beta(y)$ indéfini. Ensuite on effectue des modifications itérativement de β et de α comme suit :

1. Pour chaque y choisir l'élément x le plus intéressant dans la liste G_y parmi les x tels que $\alpha(x) = y$. et poser $\beta(y) = x$
2. Pour chaque x,
   - S'il existe y tel que $\beta(y) = x$, alors $\alpha(x) = y$.
   - S'il n'existe pas de y tel que $\beta(y) = x$ et si $\alpha(x)$ n'est pas le dernier élément de la liste F_x alors remplacer $\alpha(x)$ par l'élément qui le suit dans dans la liste F_x,
   - Si aucune des deux conditions précédentes n'est remplie alors $\alpha(x)$ est indéfini.

Programmation linéaire

Problème concret

Un magasin dispose de 24 exemplaires d'un objet A et 84 exemplaires d'un objet B. Les clients achètent, ou bien un lot à 6 euros comprenant 2 exemplaires de A et 4 exemplaires de B, ou bien un lot à 8 euros comprenent 2 exemplaires de A et 8 exemplaires de B. Comment former les lots pour que le magasin réalise une recette maximale ?

Modélisation mathématique

x_1 : nombre de lots du premier type,
x_2 : nombre de lots du deuxième type.

Interprétation géométrique

Programme linéaire

Soit A une matrice $m \times n$ de réels, b et c deux vecteurs de réels de tailles m et n respectivement. Trouver des réels $x_1, x_2, \ldots, x_n$ (les variables) qui réalisent :

$$\max \sum_{j=1}^{n} c_j x_j \quad \text{(la fonction économique)}$$

tout en respectant :

$$\forall i, \quad \sum_{j=1}^{n} A_{i,j} x_j \ \leq b_i \quad \text{(les contraintes)}$$

Remarque : max ou min; $\leq$ ou $\geq$ ou $=$, peu importe.

Interprétation économique

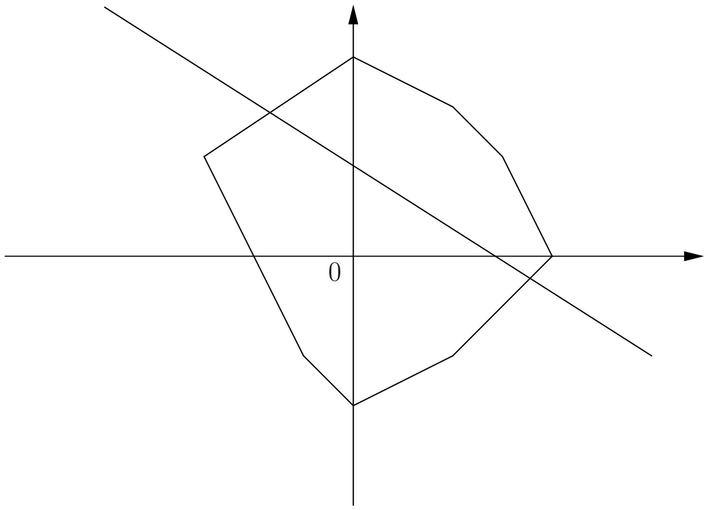
- x_j : nombre de produits finis de type j fabriqués
- c_j : bénéfice réalisé par produit fini de type j
- $a_{i,j}$: quantité de matière première de type i nécessaire pour fabriquer un produit de type j
- b_i : quantité de matière première de type i disponible

Types de situations possibles

1. Les contraintes sont impossibles à satisfaire
2. Le maximum est infini
3. Le maximum est fini et atteint par plusieurs x
4. Le maximum est fini et atteint par un x unique

Observations géométriques

- Les contraintes définissent un polyèdre convexe de $\mathbf{R}^n$
- Un sommet du polyèdre est l'intersection de n hyperplans
- Si le polyèdre est borné, alors il existe un sommet qui réalise l'optimum
- Un sommet réalise l'optimum si et seulement si ses voisins ont une valeur plus faible



Algorithme du Simplexe

Initialisation : soit P un sommet du polytope

Boucle : Tant qu'il existe un voisin Q de P qui améliore la fonction économique, faire $P \leftarrow Q$

sommet $\longleftrightarrow$ n contraintes libres : lignes I de A

sommet adjacent $\longleftrightarrow$ contraintes : lignes $I - \{h\} + \{s\}$

Exemple détaillé

sommet : intersection de deux des droites $x_1 = 0$, $x_2 = 0$, $2x_1 + 2x_2 = 24$, $4x_1 + 8x_2 = 84$.

Nouvelles variables :

$$\max 6x_1 + 8x_2$$

$$\begin{cases} 2x_1 + 2x_2 + t_1 & = & 24 \\ 4x_1 + 8x_2 + t_2 & = & 84 \\ x_1, x_2, t_1, t_2 & \geq & 0 \end{cases}$$

sommet : au moins 2 des variables x_1, x_2, t_1, t_2 sont nulles.

Initialisation : $x_1 = x_2 = 0$. Valeur 0. On a $t_1 = 24, t_2 = 84$.

Première étape

$$\max 6x_1 + 8x_2$$

$$\begin{cases} t_1 & = & 24 - 2x_1 - 2x_2 \\ t_2 & = & 84 - 4x_1 - 8x_2 \end{cases}$$

8 est positif $\Rightarrow$ si on augmente x_2 tout en laissant $x_1 = 0$, la fonction économique devient meilleure.

Jusqu'où ? t_1 devient nul à $x_2 = 24/2 = 12$, et t_2 devient nul à $84/8 = 10.5$, donc on peut augmenter x_2 jusqu'à 10.5.

Nouveau sommet : $x_1 = 0, x_2 = 10.5, t_1 = 3, t_2 = 0$, valeur 84.

Deuxième étape

On réécrit la fonction économique comme fonction de x_1, t_2 et les contraintes d'égalité en exprimant x_2, t_1 en fonction de x_1, t_2.

$$\max 84 + 2x_1 - t_2$$

$$\begin{cases} x_2 & = & 10.5 - \frac{1}{2}x_1 - \frac{1}{8}t_2 \\ t_1 & = & 3 - x_1 + \frac{1}{4}t_2 \end{cases}$$

2 est positif $\Rightarrow$ si on augmente x_1 tout en laissant $t_2 = 0$, la fonction économique devient meilleure.

Jusqu'où ? x_2 devient nul à $x_1 = 21$, et t_1 devient nul à $x_1 = 3$, donc on peut augmenter x_1 jusqu'à 3.

Nouveau sommet : $x_1 = 3, x_2 = 9, t_1 = 0, t_2 = 0$, valeur 90.

Troisième étape

On réécrit la fonction économique comme fonction de t_1, t_2.

$$\max 90 - 2t_1 - \frac{1}{2}t_2$$

Coefficients négatifs : on ne peut augmenter ni t_1, ni t_2. Le sommet satisfait deux contraintes, aucune des deux arêtes associées ne peut augmenter la fonction économique : on est donc à l'optimum.

Résultat : $x_1 = 3, x_2 = 9$, valeur 90.

Interprétation géométrique : chemin le long des sommets et arêtes du polytope ; algorithme de type glouton.

Difficultés de l'algorithme

Initialisation : Trouver un premier sommet s'il existe ?

Si $b \geq 0$, $x = 0$ satisfait $Ax \leq b$ et $x \geq 0$.

Si $b_i \leq 0$, on crée un nouveau programme linéaire comme suit: on crée une nouvelle variable $y \geq 0$, et pour chaque i tel que $b_i < 0$, on remplace la contrainte

$$\sum_j a_{ij}x_j \leq b_i$$

par

$$\sum_j a_{ij}xj - y \leq b_i$$

et la fonction économique par $\min y$. On résoud ce nouveau programme linéaire par l'algorithme du simplexe avec pour solution initiale $x = 0$, $y = \max_i(-b_i)$. Si l'optimum est 0, on peut en déduire un premier sommet.

Difficultés de l'algorithme

Et si le maximum est infini ?

Présence d'un sommet P et d'une arête semi-infinie adjacente à P permettant d'augmenter la fonction économique.

Difficultés de l'algorithme

Et si on cycle ?

Possible dans les cas dégénérés : si un sommet satisfait plus de n contraintes avec égalité.

L'algorithme du simplexe a complexité exponentielle

On montre que le système suivant à n variables donne lieu à un nombre d'étapes voisin de 2^n :

$$\max \sum_{j=1}^{n} 10^{n-j} x_j$$

$$(2 \sum_{j=1}^{i-1} 10^{i-j} x_j) + x_i \leq 100^{i-1} \quad (i = 1, 2, \ldots, n)$$

Dualité

Programme linéaire P

$$\max 4x_1 + 6x_2 + 20x_3 + 17x_4$$

$$\begin{cases} x_1 + x_3 + 2x_4 & \leq & 10 \\ x_2 + 2x_3 + x_4 & \leq & 4 \\ x_1, x_2, x_3, x_4 & \geq & 0. \end{cases}$$

Comment déduire d'autres contraintes ? Multiplions la i-ième contrainte par $u_i \geq 0$: on en déduit

$$u_1x_1 + u_2x_2 + (u_1 + 2u_2)x_3 + (2u_1 + u_2)x_4 \leq 10u_1 + 4u_2.$$

Si les coefficients de x_1, x_2, x_3, x_4 sont supérieurs ou égaux à $4, 6, 20, 17$ respectivement, **alors** évidemment $10u_1 + 4u_2$ est une borne supérieure aux valeurs possibles de la fonction économique.

Nouveau programme linéaire D

$$\min 10u_1 + 4u_2$$

$$\begin{cases} u_1 & \geq & 4 \\ u_2 & \geq & 6 \\ u_1 + 2u_2 & \geq & 20 \\ u_1 + u_2 & \geq & 17 \\ u_1, u_2 & \geq & 0. \end{cases}$$

Par construction de D, pour tout u satisfaisant aux contraintes de D et tout x satisfaisant aux contraintes de P, on a

$$4x_1 + 6x_2 + 20x_3 + 17x_4 \leq 10u_1 + 4u_2,$$

et donc $\mathrm{OPT}_P \leq \mathrm{OPT}_D$; par conséquent, si de plus on a égalité, alors x est solution optimale de P et u est solution optimale de D.

Théorème de dualité

Si l'un des deux problèmes suivants

$$\begin{array}{ll} (\mathcal{P}) & \max\{cx \mid Ax \leq b, x \geq 0\} \\ (\mathcal{D}) & \min\{bu \mid u \geq 0, A^T u \geq c\} \end{array}$$

admet une solution finie, alors il en est de même pour l'autre et leurs valeurs sont égales.

Preuve : interprétation géométrique du dual

Combinaison linéaire positive de contraintes $\iff$ nouvelle contrainte passant par le même sommet.

si x_0 sommet optimal, alors cx_0 est la valeur optimale du programme P ; il existe une combinaison linéaire des contraintes donnant une contrainte de la forme $cx \leq cx_0$, d'où cx_0 réalisable dans le dual et donc cx_0 valeur optimale pour D.

Interprétation économique du dual

Si on augmente la quantité de matière première d'une valeur marginale

$$b_i \leftarrow b_i + \epsilon_i,$$

alors l'augmentation du gain est $y_i t_i$, où y_i est la valeur qui rend optimum le problème dual.

Méthode de l'ellipsoïde

1. Réduction à la résolution de systèmes d'inéquations linéaires.

Si on sait résoudre les systèmes d'inéquations linéaires en temps polynomial, alors on sait résoudre un programme linéaire en temps polynomial.

$$\begin{cases} \max cx \\ Ax \leq b \\ x \geq 0 \end{cases} \text{ et } \begin{cases} \min bu \\ A^T u \geq c \\ u \geq 0 \end{cases} \iff \begin{cases} Ax \leq b \\ A^T u \geq c \\ x, u \geq 0 \\ cx \geq bu \end{cases}$$

2. Résoudre un système d'inéquations.

Soit D le nombre total de bits dans l'écriture des $a_{i,j}$ et des b_i en binaire. Si le système $Ax < b$ admet une solution, alors il en existe une telle que

$$\forall j, \quad -2^D \leq x_j \leq 2^D.$$

D'autre part, le volume du polyèdre est $\geq 2^{-(n+1)D}$.

Un **ellipsoïde** E de centre c est défini par $(x-c)^t A^t A(x-c) \leq 1$. Un demi-ellipsoïde s'obtient en intersectant E avec un demi-espace limité par un hyperplan qui passant par c : $hx \geq hc$. Tout demi-ellipsoïde est contenu dans un ellipsoïde E' tel que

$$\mathrm{Vol}(E') \leq e^{-\frac{1}{2(n+1)}} \mathrm{Vol}(E).$$

Principe de l'algorithme de l'ellipsoïde

Initialisation : $E \leftarrow$ la sphère $\sum_{i=1}^{n} x_i^2 \leq n4^D$.

Boucle : Tant que $c_E \notin P$ et $\mathrm{Vol}(E) \geq 2^{-(n+1)D}$, faire :

- Soit h une contrainte non satisfaite par c_E ; intersecter E avec le demi-espace défini par la contrainte $\Rightarrow D$.
- Remplacer E par l'ellipsoïde E' contenant D

Sortie : si $c_E \in P$, c'est une solution ; sinon, il n'y en a pas

Complexité : polynomiale.

Problèmes NP-Complets

1. Modèles de calcul, complexité.
2. $\mathcal{NP}$-complétude
3. Problèmes logiques $\mathcal{NP}$ complets
4. Problèmes de graphes $\mathcal{NP}$ complets
5. Problèmes en nombres entiers $\mathcal{NP}$ complets

Qu'est-ce qu'un calcul ?

Problème : défini par les données d'entrée, et pour chaque donnée, le résultat désiré. (= spécification).

1. **Machine de Turing :** Modélise un programme. La mémoire consiste en un ruban sur lequel sont écrits des 0 et des 1, un curseur se déplace le long du ruban et, en fonction de l' "état" courant de la machine, modifie le contenu de la mémoire, l'état courant, et déplace éventuellement le curseur d'une case vers la droite ou vers la gauche. Donné d'entrée: écrite sur le ruban.
2. **Programme d'un mini-langage de base :** Types autorisés : caractères, chaînes de caractères. Opérations autorisées: comparaison au mot vide, concaténation d'un caractère, accès à ou suppression du premier caractère. Instructions autorisées : affectation de caractères, conditionnelle `if...then...else...` ou `case...`, itération `while...do...`.

Processus : pour une entrée donnée, suite des instructions exécutées lors du calcul.

Problème des boucles infinies.

Indécidabilité : Il existe des problèmes pour lesquels il n'existe pas de programme qui termine toujours au bout d'un temps fini.

Exemple : Correspondance de Post.
Donnée : couples de mots

$$(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n).$$

Résultat : existe-t-il un mot w tel que

$$w = u_{i_1} u_{i_2} \ldots u_{i_q} = v_{i_1} v_{i_2} \ldots v_{i_q}?$$

Thèse de Church : Tous les modèles de calcul effectif ont la même puissance et sont traductibles l'un par l'autre.

Complexité

Dans ce cours, tous les problèmes étudiés sont décidables.

Complexité : Un algorithme a complexité polynomiale s'il existe un polynome p tel que le nombre d'instructions exécutées par le processus soit au plus $p(n)$, où le maximum est pris sur toutes les données d'entrée de taille n. Un problème a complexité polynomiale s'il existe un algorithme polynomial pour le résoudre.

Postulat : Tous les modèle de calcul raisonnables sont équivalents du point de vue de la complexité polynomiale.

Postulat : Un algorithme est efficace s'il a complexité polynomiale.

La grande question : identifier les problèmes de complexité polynomiale.

Classe $\mathcal{P}$

Problème de décision : Résultat binaire $\in$ {"oui","non"}, réponse à la question : La donné d'entrée possède-t-elle la propriété suivante : ... ?

L'ensemble des **problèmes de décision polynomiaux** constitue la classe $\mathcal{P}$.

Exemples : existence d'un arbre couvrant de poids $\leq p$, d'un chemin de longueur $\leq \ell$ entre deux sommets d'un graphe, d'un flot de valeur $\geq v$, d'une coupe de capacité $\leq c$, d'un couplage de taille $\geq t$, d'une solution faisable pour un programme linéaire de valeur $\leq v$, ...

Classe $\mathcal{NP}$

Un problème de décision Π est dans la **classe $\mathcal{NP}$** s'il existe une propriété Π' de $\mathcal{P}$ et un polynome p tel que : pour toute entrée w du problème Π, la réponse est "oui" si et seulement s'il existe un mot v de longueur au plus $p(|w|)$ tel que (w, v) satisfasse la propriété Π'.

$$w \in \textbf{Oui}(\Pi) \iff \exists v \textbf{ tel que } (|v| \leq p(|w|) \textbf{ et } (w, v) \in \textbf{Oui}(\Pi')).$$

Exemple : chemin hamiltonien

Problème Π. Entrée w : graphe non orienté. Propriété : existe-t-il un chemin qui passe une fois et une seule par tous les sommets ?

Problème Π'. Entrée (w, v): w graphe non orienté, v suite de n sommets du graphe. Résultat : v est-il un chemin de w qui passe une fois et une seule par tous les sommets ?

$\Pi' \in \mathcal{P}$ **:** évident. $|v| \leq p(|w|)$ **:** clair.

w possède un chemin hamiltonien si et seulement s'il existe un v tel que v soit un chemin hamiltonien de w.

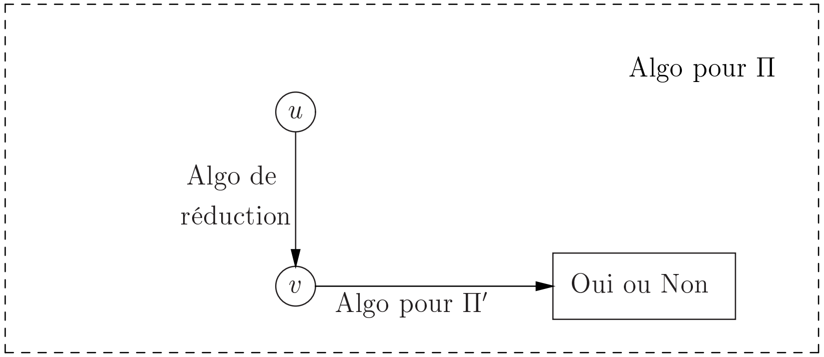
Remarque sur $\mathcal{NP}$: Oracle qui donne la réponse v, lire cette réponse et vérifier qu'elle est correcte prend un temps polynomial.

LA conjecture de l'informatique théorique

$\mathcal{P} \neq \mathcal{NP}$?

Réduction polynomiale

Un problème Π **se réduit à** un problème Π' s'il existe un algorithme polynomial qui transforme une entrée u de Π en une entrée v de Π' telle que : $u \in \Pi' \iff v \in \Pi$.



$\mathcal{NP}$ complétude

- Si $\Pi' \in \mathcal{P}$ alors $\Pi \in \mathcal{P}$.
- Si $\Pi \notin \mathcal{P}$ alors $\Pi' \notin \mathcal{P}$.
- La réduction est transitive.

Définition : Un problème Π' de $\mathcal{NP}$ est **$\mathcal{NP}$ complet** si pour tout problème Π de $\mathcal{NP}$, il existe une réduction polynomiale de Π à Π'.

Propriété : si Π est $\mathcal{NP}$ complet et si $\Pi \in \mathcal{P}$, alors $\mathcal{P} = \mathcal{NP}$.

Propriété bis : si Π est $\mathcal{NP}$ complet et si $\mathcal{P} \neq \mathcal{NP}$, alors il n'existe pas d'algorithme polynomial pour Π.

SAT : un premier problème $\mathcal{NP}$ complet

- **SAT :** Entrée : formule CNF, $(x_1 \vee \bar{x_2}) \wedge (x_2 \vee x_4 \vee \bar{x_7}) \wedge \ldots$.
  Résultat : existe-t-il une assignation de valeurs de vérité aux variables x_i qui rende la formule vraie ?
- **SAT $\in \mathcal{NP}$:** on devine l'assignation et on vérifie que toutes les clauses sont satisfaites.
- **Théorème de Cook :** Soit $\Pi \in \mathcal{NP}$. Il existe un polynôme p et un algorithme qui pour chaque entrée w de Π, construit en temps $p(|w|)$ une expression booléenne f_w telle que

$$w \in \Pi \Leftrightarrow f_w \text{ satisfaisable}$$

- **Preuve :** voir cours Majeure 1.

Comment trouver d'autres problèmes $\mathcal{NP}$ complets ?

Soit $Q \in \mathcal{NP}$. Supposons qu'on connaisse déjà un problème $\mathcal{NP}$ complet Π. Si Π se réduit polynomialement à Q, alors :

Q est $\mathcal{NP}$ complet.

3-SAT est $\mathcal{NP}$ complet

3-SAT : les clauses contiennent au plus trois littéraux

3-SAT $\in \mathcal{NP}$: Deviner l'assignation et vérifier qu'elle rend la formule vraie.

SAT se réduit à 3-SAT : On prend les clauses de la formule SAT une à une et on les remplace par des clauses à au plus 3 littéraux.

$$C = x \vee y \vee z \vee u \vee v \vee w \vee t.$$

On introduit les nouvelles variables a, b, c et on remplace C par

$$(x \vee y \vee a) \wedge (\bar{a} \vee u \vee b) \wedge (\bar{b} \vee v \vee c) \wedge (\bar{c} \vee w \vee t).$$

Remarque : 2-SAT $\in \mathcal{P}$

On réécrit le problème sous forme de recherche de chemins dans le graphe orienté G suivant :

- Pour chaque variable x_i, G a deux sommets étiquetés x_i et $\bar{x}_i$.
- Pour chaque clause $(y_i \vee y_j)$, G a un arc $\overline{y_i} \to y_j$ et un arc $\overline{y_j} \to y_i$, traduisant les implications "si y_i est faux alors y_j doit être vrai" et "si y_j est faux alors y_i doit être vrai".
- On montre que la formule est satisfaisable si et seulement si, dans le graphe G, pour aucun i il n'existe de circuit passant à la fois par y_i et par $\overline{y_i}$.

STABLE est $\mathcal{NP}$ complet

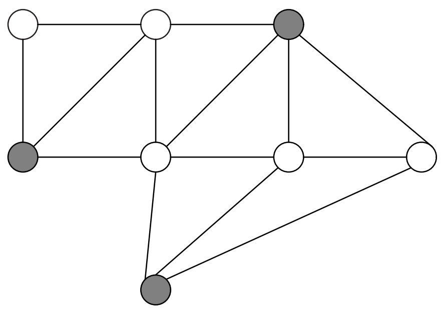


Figure 8: Un graphe et un de ses stables

Un **stable** est un sous-ensemble S de V tel que deux sommets de S ne sont pas joints par une arête.

Le problème STABLE : Entrée : (G, k). Existence d'un stable de taille $\geq k$? STABLE $\in \mathcal{NP}$: clair.

Réduction de 3-SAT à STABLE

Soit f une formule comportant n clauses chacune avec exactement trois littéraux. On construit le graphe G ayant $3n$ sommets. On relie par trois arêtes les sommets correspondants à une clause. On relie par une arête tous les couples de sommets qui correspondent à un x_i et un $\overline{x_i}$. Alors : f est satisfaisable si et seulement si G possède un indépendant de taille n.

COUVRANT-MIN est $\mathcal{NP}$ complet

Un **ensemble couvrant** Y de sommets est un sous-ensemble de V tel que toute arête du graphe possède au moins une extremité dans Y.

Idée : Un ensemble couvrant est le complémentaire d'un indépendant.

SUBSET-SUM est $\mathcal{NP}$ complet

Données d'entrée : Un ensemble fini d'entiers S, et un entier x.

Problème : Existe-t-il $F \subset S$ tel que $\sum_{f\in F} f = x$?

SUBSET-SUM $\in \mathcal{NP}$: clair.

Réduction : A partir de COUVRANT-MIN.

COUVRANT-MIN se réduit à SUBSET-SUM

Soit $G = (V, E)$ le graphe d'entrée de COUVRANT-MIN. Soient $x_1, \ldots, x_n$ ses sommets et $e_0, e_1, \ldots, e_{m-1}$ ses arêtes. On veut construire un ensemble S et un entier x :

- A chaque sommet x_i incident à $e_{i_1}, e_{i_2}, \ldots, e_{i_p}$ on associe le nombre
$$a_i = \sum_{j=1}^{p} 4^{i_j} + 4^m.$$
- $S = \{a_1, a_2, \ldots, a_n\} \cup \{1, 4, 4^2, \ldots, 4^{m-1}\}$
- $x = 2\left(\sum_{j=0}^{m-1} 4^j\right) + k4^m.$

Analyse

x est somme d'un sous-ensemble de S ssi il existe un ensemble couvrant de sommets de taille k pour G :

Tout entier x qui est somme des éléments de S s'écrit (en base 4) :

$$x = \sum_{j=0}^{m-1} u_i 4^j \quad + \quad y4^m \quad \text{avec} \quad 0 \leq u_i \leq 3.$$

Traiter les problèmes NP-Complets

1. Approximations : Sac à dos, sommets couvrants, bin packing
2. Bornes inférieures : Bin packing
3. Schémas d'approximation : Sac à dos
4. Heuristiques : Backtracking, Branch and Bound

Sac à dos

Problème : Etant donné un poids maximum P et n objets, où l'objet i a poids $p_i \leq P$ et valeur a_i, trouver $I \subset \{1, 2, \ldots, n\}$ qui maximise $\sum_{i \in I} a_i$ sous la contrainte $\sum_{i \in I} p_i \leq P$.

Objectif équivalent : Trouver $0 \leq x_i \leq 1$ entiers qui maximisent $\sum x_i a_i$ sous la contrainte $\sum x_i p_i \leq P$.

Rappel : Décider si on peut arriver à un bénéfice $\geq A$ est un problème $\mathcal{NP}$ complet.

Solution pour objets fractionnaires (x_i réels)

Supposons $a_1/p_1 \geq a_2/p_2 \geq \ldots \geq a_n/p_n$.

Initialisation : sac vide; $j \leftarrow 1$;

Boucle : Tant que j tient dans le sac

{ ajouter j dans le sac ; $j \leftarrow j + 1$; }

Objet j : en mettre une fraction dans le sac pour le remplir exactement

Bénéfice réalisé : $\sum_{k<j} a_k + x_j a_j$, avec $0 \leq x_j < 1$.

Algorithme d'approximation (x_i entiers)

On choisit la meilleure des deux solutions suivantes.

1. Supposons $a_1/p_1 \geq a_2/p_2 \geq \ldots \geq a_n/p_n$.

Initialisation : sac vide; $j \leftarrow 1$;

Boucle : Tant que j tient dans le sac

{ ajouter j dans le sac; $j \leftarrow j+1$; }

Bénéfice réalisé : $\sum_{k<j} a_k$.

2. Mettre uniquement dans le sac l'objet de bénéfice maximum.

Bénéfice réalisé : $\max_k a_k$.

Analyse

Bénéfice réalisé :

$$\max(\sum_{k<j} a_k, \max_k a_k) \geq \frac{1}{2}(\sum_{k<j} a_k + \max_k a_k) \geq \frac{1}{2}(\sum_{k<j} a_k + a_j) \geq \frac{1}{2}(\sum_{k<j} a_k + x_j a_j).$$

Mais la solution optimale dans le cas fractionnaire est **au moins aussi bonne** que la solution optimale du cas entier.

Théorème L'algorithme donne une solution de bénéfice égal à au moins la moitié du bénéfice optimal.

COUVRANT-MIN

Problème : Recouvrir toutes les arêtes d'un graphe avec le nombre minimum de sommets.

Rappel : Décider s'il y a une couverture de cardinal $\leq k$ est $\mathcal{NP}$ complet.

Algorithme d'approximation

Initialisation : $Y \leftarrow \emptyset$;

Boucle : Tant qu'il existe une arête $e = \{u, v\}$ non couverte par Y :

ajouter u et v à Y

Analyse

Soit F la suite des arêtes choisies lors de la boucle. L'ensemble couvrant construit par l'algorithme a cardinal $2|F|$.

Soit C un ensemble couvrant de cardinal minimum. Pour tout $e \in F$, au moins une de ses extrémités est dans C. Mais les arêtes de F sont disjointes (couplage). Donc $|C| \geq |F|$.

Théorème L'algorithme donne une solution de cardinal au plus double du cardinal optimal.

BIN PACKING

Problème : Ranger n objets de poids $p_1, \ldots, p_n$ un nombre minimum de boites de capacité P.

Entrée :

$$6.1,\ 4.1,\ 4,\ 4,\ 2,\ 1.9,\ 1.8, \qquad P = 12$$

Résultat :

$$6.1 + 4 + 1.9 = 12, \qquad 4.1 + 4 + 2 + 1.8 = 11.9.$$

FIRST FIT DECREASING (FFD)

Supposons $p_1 \geq p_2 \geq \ldots \geq p_n$.

Boucle : Pour i allant de 1 à n faire

{ $j \leftarrow 1$;

Tant que p_i ne tient pas dans B_j faire $j \leftarrow j+1$;

Mettre p_i dans B_j; }

Sur l'exemple : $6.1 + 4.1; \quad 4 + 4 + 2 + 1.9; \quad 1.8$

Théorème FIRST FIT DECREASING utilise $1 + 3$ OPT$/2$ boîtes au maximum.

Preuve.

Cas 1. Si tous les poids sont $> \frac{P}{3}$, alors FFD est optimal.

$$\underbrace{p_1, p_2, \ldots, p_i,}_{>2P/3} \quad \underbrace{p_{i+1} \ldots, p_j,}_{2P/3 \geq .. > P/2} \quad \underbrace{p_{j+1}, \ldots, p_n}_{P/2 \geq .. > P/3}$$

Cas 2. Si par FFD chaque boîte contient un objet de poids $> \frac{P}{3}$, alors FFD est optimal.

Pour le voir, supprimer les objets de poids $\leq P/3$ pour se ramener au premier cas.

Cas 3. S'il existe une boîte ne contenant que des objets de poids $\leq P/3$, alors toutes les autres boîtes sont remplies au moins aux $2/3$.

Par FFD, la dernière boîte k est alors une boîte ne contenant que des objets de poids $\leq P/3$.

Comment cette boîte a-t-elle été ouverte ? Lors de l'insertion d'un objet de poids $\leq P/3$, et donc nécessairement parce que toutes les boîtes précédentes, $[1 \ldots k-1]$, étaient déjà toutes pleines aux $2/3$.

Le poids total des objets placés est donc au moins

$$(k-1)2P/3 \leq \text{OPT}.$$

Remarque : L'analyse peut être améliorée $\Rightarrow 11/9$ OPT.

Schémas d'approximation

Problème : optimiser une fonction f sous les contraintes $x \in X$.

Un **schéma d'approximation** consiste en la donnée, pour tout $\varepsilon > 0$, d'un algorithme A_ε qui pour toute instance du problème trouve un x satisfaisant les contraintes ($x \in X$) et tel que:

$$\frac{|f(x) - \text{OPT}|}{\text{OPT}} \leq \varepsilon.$$

Non-approximabilité du BIN PACKING

Théorème S'il existe un algorithme d'approximation pour BIN PACKING avec erreur relative $\varepsilon = 1/3$, alors $P = NP$.

Réduction à partir de SUBSET SUM. Soit $(E = \{f_1, \ldots, f_n\}, x)$ la donnée d'entrée, avec $x \leq \sum_i f_i/2$. Soit $f = \sum_i f_i$.

$$\sum_{i \in F} f_i = x \iff \sum_{i \in F} f_i + 2f = \sum_{i \notin F} f_i + f + 2x = 2f + x.$$

Réduction : Donnée d'entrées du problème de BIN PACKING : objets $f_1, \ldots, f_n, f + 2x, 2f$, capacité $2f + x$.

Analyse : Instance positive de SUBSET SUM ssi 2 boîtes suffisent pour le BIN PACKING.

Conclusion. Tout le problème est de savoir si 2 boîtes suffisent ou si 3 boîtes sont nécessaires. Cette distinction est fait par un algorithme d'approximation avec $\varepsilon = 1/3$.

Rappels sur le SAC A DOS

Problème : On cherche les $x_i \in \{0, 1\}$ qui maximisent $\sum_{i=1}^n a_i x_i$ sous la contrainte $\sum_{i=1}^n p_i x_i \leq p$.

Problème $\mathcal{NP}$ **difficile** en général.

On suppose que tous les p_i sont $\leq p$. On utilise une variante du **programme dynamique** du cours 2 : soit $P[i, v]$ le plus petit poids réalisable par choix d'objets dans $\{1, \ldots i\}$ garantissant valeur v.

$$\begin{cases} P[0, v] &= +\infty \\ P[i, v] &= \min\{P[i-1, v], P[i-1, v-a_i] + p_i\} \end{cases}$$

Complexité $O(n \sum a_i)$, **polynomiale** si les a_i sont de petits entiers.

Un schéma d'approximation pour SAC A DOS

Idée de l'algorithme : Arrondir les valeurs des objets. Soit

$$k = \lceil \log_{10}(\varepsilon \sum a_i / n^2) \rceil.$$

On remplace les valeurs a_i par $b_i = \lfloor a_i / 10^k \rfloor$, et on résoud par programmation dynamique.

Analyse de complexité : $O(n \times \sum a_i / 10^k) = O(n^3/\varepsilon)$, polynomiale.

Analyse de l'erreur : OPT vaut au moins $\max_i a_i \geq \sum a_i / n$. Si on borne l'erreur absolue par $n 10^k$, alors ceci est borné par $\varepsilon \sum a_i / n$, et alors l'erreur relative sera d'au plus ε.

Analyse de l'erreur absolue

Soit x_i la solution optimale et x'_i la solution obtenue, optimale pour le problème arrondi. Ainsi :

$$\sum a_i x'_i \leq \sum a_i x_i \text{ et } \sum b_i x_i \leq \sum b_i x'_i.$$

Par définition des b_i :

$$\sum 10^k \lfloor \frac{a_i}{10^k} \rfloor x_i \leq \sum 10^k \lfloor \frac{a_i}{10^k} \rfloor x'_i.$$

Donc

$$\sum (a_i - 10^k) x_i \leq \sum a_i x'_i.$$

Erreur absolue :

$$|\sum a_i x'_i - \sum a_i x_i| \leq \sum 10^k x_i \leq n 10^k.$$

Backtracking, exploration arborescente

- Un problème de décision à résoudre (exemple SAT)
- On part d'un ensemble S de problèmes initialisé par celui à resoudre
- On choisit (astucieusement!) un des problèmes P que l'on supprime de S
- On décompose ce problème en plusieurs problèmes $P_1, P_2, \ldots, P_k$ formant une liste L
- On a ainsi:

$$(\exists x, P(x)) \Leftrightarrow \vee_{i=1,\ldots k} (\exists x, P_i(x))$$

Backtracking, suite

- Si l'un des problèmes de L a une solution immédiate on la donne
- Sinon on supprime les problèmes de L qui de manière évidente n'ont pas de solution et on ajoute ceux qui restent dans S
- On termine quand S est vide ou quand on a trouvé une solution.
- On parcourt un arbre de solutions exhaustivement
- Prolog, programmation par contraintes

Exemple 3-SAT

- Choisir un problème qui contient des clauses a une ou 2 variables s'il y en a
- Choisir un problème dont le nombre de clauses est le plus petit
- Donner les valeurs vrai et faux à une variable qui intervient dans une clause à 1 ou 2 variables. Síl n'y en a pas prendre celle qui apparaît le plus grand nombre de fois dans une clause à trois variables.
- Chaque problème se divise en 2, certains pouvant être simplement satisfaits ou par contre trivialement insatisfaisables.

Branch and Bound

- On a un problème d'optimisation
- On connait une valeur satisfaisant aux contraintes
- On n'explore pas les sous-arbres dont on peut dire à priori qu'ils ne donneront pas une valeur plus intéressante que la valeur déjà trouvée

Exemple programmation en nombre entiers

- On résout le problème en supprimant la contrainte solutions en nombre entiers
- On chosit une variable x_i qui dans la solution optimale est affectée d'une valeur α_i non entière
- On crée deux problèmes avec plus de contraintes, à l'un on ajoute $x_i \leq \lfloor \alpha_i \rfloor$ à l'autre $x_i \geq \lfloor \alpha_i \rfloor + 1$

Algorithmes Probabilistes

1. Généralités
2. propriété de Ramsey : la “méthode probabiliste"
3. Programmation linéaire en petite dimension
4. Hachage parfait
5. Routage sur l'hypercube : permutation aléatoire
6. Exclusion mutuelle : les philosophes

Algorithme Probabiliste

Lors de l'exécution, l'algorithme fait des choix probabilistes guidés par des tirages aléatoires : lancers de pièces pile ou face.

Deux catégories

1. Monte Carlo Le résultat du calcul est aléatoire. Toujours rapide, Probablement correct
2. Las Vegas Le temps d'exécution est aléatoire. Toujours correct, Probablement rapide

Outils d'analyse probabiliste

Inégalité de Markov. Soit X une variable aléatoire positive de moyenne $E(X)$. Alors pour tout t :

$$Pr(X \geq t \cdot E(X)) \leq \frac{1}{t}.$$

Inégalité de Chebychev : Soit X une variable aléatoire X de moyenne μ et d'écart-type σ. Alors pour tout a :

$$Pr(|X - \mu| \geq a\sigma) \leq \frac{1}{a^2}.$$

Grandes déviations Soient $X_1, \ldots, X_n$ n variables aléatoires indépendantes valant 1 avec probabilité p et 0 avec probabilité $1-p$, et soit $X = \sum X_i$, $E(X) = np$. Alors :

$$Pr(X \geq (1+\delta)np) < \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^{np}.$$

Analyse probabiliste ou algorithme probabiliste

Analyse de quicksort : Complexité $\Theta(n^2)$ dans le pire cas, mais $O(n \log n)$ en moyenne si les nombres à trier forment une permutation aléatoire.

Quicksort randomisé :

1. Faire une permutation aléatoire des nombres donnés en entrée, puis
2. Exécuter Quicksort

Analyse de quicksort randomisé : La complexité est une variable aléatoire. Quels que soient les nombres donnés en entrée, la complexité est $O(n \log n)$ en moyenne, où la moyenne porte sur le choix de la permutation aléatoire.

Avantages : contrôle de la distribution, donc pas de données pathologiques

Propriété de Ramsey

Dans tout groupe de **n=6** personnes, il existe toujours, ou bien **k=3** personnes qui se connaissent, ou bien **3** personnes qui ne se connaissent pas.

Faux pour les groupes de **n=5** personnes.

Quelle doit être la taille du groupe pour garantir l'existence de **k=4** personnes se connaissant toutes mutuellement ou ne se connaissant pas du tout mutuellement ? **n=17**.

Théorème : Si n est tel que

$$2\binom{n}{k} < 2^{k(k-1)/2},$$

alors il existe des groupes de n personnes sans sous-groupe de k personnes qui se connaissent toutes mutuellement ou ne se connaissent pas du tout mutuellement.

Preuve :

1. On prend une configuration **aléatoire** de n personnes. $\Pr\{i \text{ connaît } j\} = 1/2$.
2. On montre que

   $$\Pr\{\exists \text{ sous-groupe de } k \text{ personnes satisfaisant la propriété }\} < 1.$$

3. On en déduit qu'il existe une configuration démontrant le théorème.

Problème ouvert : construction explicite.

Programmation linéaire en petite dimension

Problème : n contraintes linéaires (ensemble H), d variables, d petit.

Hypothèse simplificatrices : Sans perte de généralité, on peut supposer que :

- La fonction économique est x_1,
- L'optimum est atteint en un sommet unique du polyèdre, et
- Ce sommet est défini par exactement d contraintes : ensemble S.

Algorithme

Cas de base : si n petit, utiliser l'algorithme du simplexe.

Cas général :

Initialisation : $S \leftarrow \emptyset$

Répéter

Soit R un ensemble **aléatoire** de r contraintes de $H \setminus S$

Résoudre récursivement le problème pour $R \cup S \Rightarrow$ sommet x

Soit V l'ensemble des contraintes violées par x

(Si $|V|$ pas trop grand alors) ajouter V à S

Jusqu'à ce que x satisfasse toutes les contraintes

$r = \min\{d\sqrt{n}, |H \setminus S|\}$; "pas trop grand" : $|V| < 2\sqrt{n}$.

Analyse

- Un passage dans la boucle est **satisfaisant** si $|V|$ n'est pas trop grand.
- A chaque passage satisaisant dans la boucle, S s'enrichit d'au moins une contrainte critique $\Rightarrow$ au plus d passages satisfaisants dans la boucle.
- Chaque passage dans la boucle coûte $O(dn)$ en dehors des appels récursifs
- Les appels récursifs portent sur les ensembles de contraintes de taille au plus $2d\sqrt{n}$.
- Supposons que la probabilité qu'un passage soit satisaisant soit $\geq 1/2$. Alors le nombre moyen total de passages dans la boucle est $\leq 2\sqrt{n}$.

$$T(n) \leq 2dT(2d\sqrt{n}) + O(d^2 n) \Rightarrow T(n) = O(d^2 n + (\log n)^{f(d)}).$$

Probabilité qu'un passage soit satisfaisant

On montre $E(|V|) \leq d(n-r+1)/(r-d) \leq \sqrt{n}$ et on utilise l'inégalité de Markov.

Hachage parfait

Etant donné un univers $U = \{1, 2, \ldots, N\}$ de *clés* et un ensemble X de n clés, on cherche une application injective $X \longrightarrow \{1, 2, \ldots, m\}$ de la forme :

$$h_k(x) \quad = \quad (kx \bmod N) \bmod m,$$

où k est un entier à choisir compris entre 1 et $N - 1$

Collisions

A chaque fonction h_k on associe sa mesure de collision : soit $c(i, k)$ le nombre de clés d'image i par h_k. Soit

$$\gamma(k) = \sum_{i=0}^{m-1} c(i, k)^2 \quad - n.$$

Alors h_k est une injection si et seulement si

$$\gamma(k) = 0$$

Lemme : Si N est premier alors on a :

$$\sum_k \gamma(k) \quad \leq \quad \frac{2n(n-1)(N-2)}{m}$$

Conséquence : Si k est tiré au hasard, $E(\gamma(k)) \leq 2n(n-1)/m$, et donc par Markov :

$$Pr(\gamma(k) \leq \frac{4n(n-1)}{m}) \quad > \quad \frac{1}{2}.$$

Algorithme

- `Repeter`
  - Choisir k au hasard dans $1, \ldots N - 1$
  - jusqu'à $\gamma(k) < \frac{4n(n-1)}{m}$
- Le temps moyen de calcul est $2n$
- On trouve en moyenne le résultat après deux essais

Routage sur l'hypercube

Dans un réseau représenté par un graphe $G = (X, E)$, chaque sommet i détient un message M_i qu'il doit envoyer à un autre sommet $\sigma(i)$

- Un algorithme de routage consiste à choisir à chaque étape un certain nombres de messages qui vont passer d'un sommet à l'un de ses voisins
- Au cours d'une étape deux messages ne peuvent pas emprunter la même arête
- Il s'agit de minimiser le nombre d'étapes
- On suppose que σ est une permutation (pour simplifier les calculs).

Graphe de l'Hypercube

- Les sommets sont toutes les suites de n bits
- Une arête entre chaque couple de sommets qui diffèrent sur 1 bit

Algorithme déterministe

- A chaque étape un message cherche à se rapprocher de son but en modifiant le bit le plus à gauche du sommet qui diffère de celui du but
- Si deux messages veulent emprunter la même arête on donne priorité à celui qui attend depuis le plus long temps
- Les ex aequo sont départagés au hasard

Algorithme probabiliste

1. **Initialisation** Chaque sommet détermine au hasard une destination intermédiaire $\tau(i)$
2. **Phase 1** On applique l'algorithme déterministe pour aller de chaque i à $\tau(i)$
3. **Phase 2** On applique l'algorithme déterministe pour aller de chaque $\tau(i)$ à $\sigma(i)$

Lemme 1 Deux chemins déterminés par l'algorithme déterministe se rencontrent sur une partie constituée d'arêtes consécutives

Lemme 2 Pour une phase le retard accumulé par chaque message est inférieur ou égal au nombre de messages différents qui empruntent une même arête que lui.

Nombre moyen d'étapes

- $H_{i,j}$ nombre de chemins contenant l'arête (i, j)
- Le nombre total d'arêtes est $Nn = 2^n n$
- Le nombre moyen de chemins empruntant une arête donnée est 1/2
- La valeur moyenne de la somme des $H_{i,j}$ est $n/2$.
- La probabilité que cette variable dépasse $(\theta + 1)n$ est au plus $2^{-\theta n}$
- Avec probabilité $\frac{2}{N^\theta}$ chaque message atteint sa destination finale en plus de $(2\theta + 4)$ étapes.

Le problème des philosophes

- Des philospohes sont assis autour d'une table, une assiette de spaghetti devant eux
- Chacun partage une forurchette avace son voisin de droite et une autre avec son voisin de gauche
- Il peut, vérifier si une fourchette est libre et s'en saisir, ou en relacher une
- Il mange s'il détient une fourchette dans chaque main

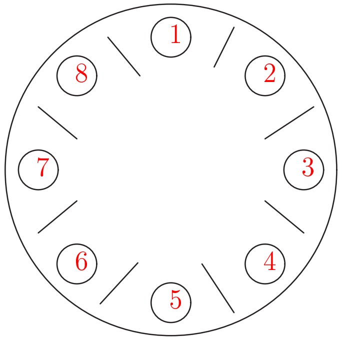


Figure 9: * philosophes autour d'une table

Problème

Trouver un algorithme pour chaque philosophe tel que :

- Deux philosophes ne détiennent pas ensemble la même fourchette Correction
- Chaque philosophe mange au bout d'un temps raisonable vivacité
- Pas de solution symétrique!

Une solution incorrecte

1. Attendre que la fourchette de gauche soit libre, la saisir
2. Attendre que la fourchette de droite soit libre, la saisir
3. Manger
4. Reposer les deux fourchettes

Modélisation

1. Les fourchettes sont des variables partagées $f[i]$ accessibles par i et $i+1$
2. Les états : Repos, Reposer-gauche, Reposer-droite, Critique, Laisser-gauche, Laisser-droite
3. Transition : changement d'état, modification des variables partagées
4. Modèle avec des automates

Algorithme Pair-impair

Pincipe intuitif :

Les philosophes de numéro pair se saisissent d'abord de la fourchette de gauche, ceux de numéro impair de la fourchette de droite

- Les variables partagées sont des files de priorité pouvant contenir deux valeurs
- Un philosophe ne prend une fourchette que s'il est premier sur sa file de priorité
- Une liste d'attente est une suite de philosophes dont chacun d'entre-eux détient une fourchette et attend que celui qui le précède dans la liste repose la sienne
- On vérifie que dans l'algorithme pair/impair, les listes d'attentes sont au plus de taille 2
- Une file de priorité correspond a une fourchette classée première pour deux philosophes

Algorithme probabiliste

Pincipe intuitif :

Chaque philosphe tire au sort la fourchette dont il va se saisir en premier

- Il existe des situations de blocage mais qui interviennent avec une probabilité faible
- Si ℓ est le temps maximum d'exécution d'une étape, on montre que la probabilité pour que l'on arrive en temps 14ℓ à un état dans lequel au moins un philosophe mange est supérieur ou égale à

$$\frac{1}{16}$$

Algorithme

1. **Variables partagées** `f[i]` initialisée à `false` accessible en lecture-écriture par les processus i et $i-1$
2. **Algorithme du processus i :**

   `do forever`
   - `first := random(0,1);`
   - `wait until f[first] = false`
   - `if f[1-first] = false then`
     - `f[1-first] = true`
     - `go to` Crit
   - `else f[first] := false`

Preuve de validité

- On note $\mathcal{T}$ l'ensemble des états où l'un des processus est dans `Try`, et $\mathcal{C}$ l'ensemble des états où l'un des processus est dans la section critique
- On note

$$\mathcal{U} \xrightarrow{t}_p \mathcal{V}$$

  si on passe d'un état de $\mathcal{U}$ à un état de $\mathcal{V}$ en moins de t étapes avec une probabilité supérieure ou égale à p.
- On va montrer :

$$\mathcal{T} \xrightarrow{14\ell}_{1/16} \mathcal{C}$$

Notations

- On note $\mathcal{G}$ l'ensemble des états où l'un des processus a choisi en premier la gauche et où un de ses voisins a fait un premier choix qui n'est pas de la même fourchette.
- On note : $\mathcal{RT}$ l'ensemble des états où tous les processus sont dans un état de `Try`
- On note : $\mathcal{F}$ l'ensemble des états où l'un des processus est dans l'état `Flip`.

Transitions

- On vérifie

$$\mathcal{T} \xrightarrow{3\ell}_1 \mathcal{RT} \cup \mathcal{C}$$

- On a

$$\mathcal{RT} \xrightarrow{4\ell}_1 \mathcal{F} \cup \mathcal{C}$$

- On vérifie

$$\mathcal{F} \xrightarrow{3\ell}_{1/4} \mathcal{G} \cup \mathcal{C}$$

- On vérifie

$$\mathcal{G} \xrightarrow{6\ell}_{1/4} \mathcal{C}$$

Algorithmes pour la Bio-informatique

1. Alignement de familles de séquences
2. Recherche d'un motif dans un texte
3. Alignement local
4. Sous-séquences maximales
5. Reconstruction d'arbres d'évolution
6. Estimation de distance entre génomes

Alignement de familles de séquences

Les bases de données pour les protéines les regroupent habituellement en familles. Trouver la famille à laquelle appartient un nouveau gène.

Représentation de la famille: alignement de tous ses membres.

Alignement de **deux** séquences: cf chapitre 2.

Alignements de séquences

Etant donné deux mots sur l'alphabet $\Sigma = \{A, C, G, T\}$ on cherche un alignement qui optimise une fonction qui mesure la similarité

Similarité : $p : (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \to \mathbf{R}_+$

$$p(a,a) > 0, \quad p(a,b) < 0 \text{ si } b \neq a.$$

Exemple : GATGCAT et ATCGAT

G	A	T	–	G	C	A	T
–	A	T	C	G	–	A	T

Algorithme d'alignement

Entrée : $u_1 \cdots u_n$ et $v_1 \cdots v_m$ mots sur $\{A, C, G, T\}$.

Soit $s(i,j)$ valeur du meilleur alignement entre $u_1 \cdots u_i$ et $v_1 \cdots v_j$.

Définition récursive :

$$\begin{aligned} s(i,0) &= \sum_{k \leq i} p(u_k, -) \\ s(0,j) &= \sum_{k \leq j} p(-, v_k) \\ s(i,j) &= \max \begin{cases} s(i-1, j-1) + p(u_i, v_j) \\ s(i-1, j) + p(u_i, -) \\ s(i, j-1) + p(-, v_j) \end{cases} \end{aligned}$$

But : $s(n,m)$

Implantation : Tableau $s[0 \ldots n, 0 \ldots m]$.

Alignement de famille de séquences

Insérer des blancs dans chaque $S_i \longrightarrow S'_i$ pour obtenir $|S'_1| = |S'_2| = \ldots = |S'_k|$ en minimisant

$$\sum_{1 \leq i < j \leq k} D(i,j)$$

$$D(i,j) = \sum_{1 \leq \ell \leq N} \delta(S'_i[\ell], S'_j[\ell]).$$

NP-difficile

Un algorithme d'approximation

Centre : S_i qui minimise $\sum_j D(S_i, S_j)$, où D est calculée avec l'algorithme d'alignement de deux séquences.

1. Trouver le centre S_1.
2. Pour j allant de 2 à k faire:

   - Aligner S_i et S'_1 en utilisant l'algorithme d'alignement de deux séquences.
   - Ceci définit S'_i et modifie S'_1 en y insérant des blancs.
   - Mettre à jour $S'_2, \ldots, S'_{i-1}$ en y insérant des blancs aux mêmes endroits que dans S'_1.

Analyse

Complexité : pour k séquences de longueur n:

$$O(\frac{k(k-1)}{2}n^2 + \sum_{2\le j\le k} O(jn\cdot n)) = O(k^2n^2).$$

Qualité de la solution :

$$\sum_{1\le i<j\le k} D(S'_i,S'_j) \le \sum_{1\le i<j\le k} D(S'_i,S'_1)+D(S'_1,S'_j)$$

$$= (k-1)\sum_{1\le i\le k} D(S'_i,S'_1) = (k-1)\sum_{1\le i\le k} D(S_i,S_1).$$

Borne inférieure

Qualité optimale : Soit $S_1^*,\ldots,S_k^*$ l'alignement optimal. Sa valeur satisfait:

$$\sum_{1\le i<j\le k} D(S_i^*,S_j^*) \ge \sum_{1\le i<j\le k} D(S_i,S_j) = \frac{1}{2}\sum_i\sum_{j\ne i} D(S_i,S_j)$$

$$\ge \frac{1}{2}\sum_i\sum_j D(S_i,S_1) = \frac{k}{2}\sum_j D(S_i,S_1).$$

Conclusion : c'est une 2-approximation.

Recherche de motif

Entrée : mot P de longueur m petite (le motif), texte T de longueur n grande (le texte).

Sortie : Décider si P est un facteur de T.

Algorithme naïf : $O(mn)$

Algorithme de Knuth-Morris-Pratt : $O(m+n)$

Principe de l'algorithme

Sachant $P[1\ldots i] = T[k+1\ldots k+i]$, on compare $P[i+1]$ à $T[k+i+1]$.

Si $P[i+1] = T[k+i+1]$, parfait, on incrémente i.

Mais si $P[i+1] \ne T[k+i+1]$, on décale P vers la droite.

De combien décaler ? Le plus possible, sachant $P[1\ldots i] = T[k+1\ldots k+i]$ et $P[i+1] \ne T[k+i+1]$.

Choix du décalage

Condition nécessaire : On décale P en P'. Les x premiers caractères de P' sont les x derniers de P, le suivant est différent. x est maximum avec cette propriété.

Table s précalculée : $s(i)$ est le plus grand suffixe de $P[1 \dots i]$ qui soit aussi préfixe de P, avec $P[i+1] \neq P[s(i)+1]$.

Décalage : de $i - s(i)$ caractères vers la droite.

Analyse de complexité

Construction de s : $O(m)$ (cours Majeure 1).

Comparaisons entre P et T :

- si mismatch : on décale P vers la droite $\Longrightarrow$ au plus n telles comparaisons.
- si match : on m'emorise grâce à s et on ne fait plus jamais de comparaison à ce caractère de T $\Longrightarrow$ au plus n telles comparaisons.

Total au plus $2n$ comparaisons.

Sous-séquences maximales

Motivation : Identifier de nouveaux gènes. Le codage des gènes occupe moins de 3% du génome humain.

Idée : Les nouveaux gènes ressembleront à ceux qui sont déjà déterminés. En particulier par la distribution des codons.

Score : Chaque codon a une fréquence dans le génome et une fréquence a priori différente dans les gènes déjà connus. On se sert de ces deux fréquences pour lui associer un “score” (log-likelihood).

Idée : Une région dont le score total est élevé correspond potentiellement à un nouveau gène.

Le problème

Entrée : Suite $X_1, X_2, \dots, X_n$ de réels positifs ou négatifs, les scores.

Sortie : Facteur $X_i \dots X_j$ qui maximise $X_i + X_{i+1} + \dots + X_j$.

Comment trouver ce facteur rapidement ?

Algorithme

Soit a_k le meilleur score d'un facteur de $X_1 \ldots X_k$, et b_k le meilleur score d'un suffixe de $X_1 \ldots X_k$.

$$a_0 = b_0 = 0$$

$$\begin{aligned} a_k &= \max(a_{k-1}, b_{k-1} + X_k) \\ b_k &= \max(b_{k-1} + X_k, 0) \end{aligned}$$

Complexité : linéaire

Reconstruction d'arbre d'évolution

Entrée : n espèces; matrice $D(i, j)$: distance entre les espèces i et j et leur plus proche ancêtre commun.

Sortie : l'arbre d'évolution lorsqu'il en existe un compatible avec D.

Ultramétrique

Définition : Une matrice (n, n) définit une ultra-métrique si pour tout triplet $\{i, j, k\}$, le maximum de $D(i, j), D(j, k)$ et $D(k, i)$ est atteint par au moins deux de ces trois valeurs.

Théorème : Etant donné n espèces avec leur matrice D symétrique associée, le probléme de reconstruction de l'arbre d'évolution a une solution si et seulement si D définit une ultra-métrique.

Distance entre génomes

Entrée : Deux génomes $g = g_1 \ldots g_n$ et $g' = g_{\sigma(1)} \cdots g_{\sigma(n)}$.

Modèle de mutation : Prendre un facteur et le renverser.

$$g_1 \ldots g_i \ldots g_j \ldots g_n \to g_1 \ldots g_{i-1} g_j g_{j-1} \ldots g_i g_{j+1} \ldots g_n$$

Question : Distance entre g et g', en termes de nombre de mutations ?

Une approche stochastique

Coupure : i tel que g_i et g_{i+1}, voisins dans g, ne sont pas voisins dans g'

Facile à calculer : nombre de coupures entre g et g'.

Idée : s'en servir comme d'un estimateur de la distance.

Algorithme

Soit D le nombre de coupures entre g et g'.

Soit D_t le nombre de coupes entre $g(t)$ et $g(0)$, où $g(t)$ est le descendant de g après t mutations.

1. On calcule $E(D_t)$ pour $t = 1, 2, 3, \ldots$ en procédant par récurrence.
2. On choisit le t tel que $|E(D_t) - D|$ soit minimum.