Traiter les problèmes NP-Complets

1. Algorithmes approchés
2. Schémas d'approximation
3. Bin-packing: approximable
4. Sac à dos : très bien approximable
5. Voyageur de commerce : non approximable
6. Backtracking, Branch and Bound.

Algorithmes ε-approchés

- Trouver parmi les éléments d'un ensemble X celui qui rend maximum une fonction f de X
- On dit qu'un algorithme est ε-*approché* s'il donne un résultat x qui satisfait, (x_0 est le maximum de f):

$$\frac{f(x_0) - f(x)}{f(x_0)} < \varepsilon$$

- Dans le cas d'un algorithme de recherche d'un coût $c(x)$ minimum :

$$\frac{c(x) - c(x_0)}{c(x)} < \varepsilon$$

Algorithme $1/2$-approché pour COUVRANT-MIN

```
F = E; Y = ∅;
Tant que (F n'est pas vide) {
        Choisir une arete quelconque f dans F;
        Ajouter dans Y les deux extremites x et y de f;
        Supprimer de F toute arete
                    dont une des extremites est x ou y.
        }
```

- On vérifie facilement que si un ensemble C d'arêtes constitue un couplage, tout ensemble couvrant contient une des extrémités de chaque arête C
- $|C|$ est un minorant pour le cardinal d'un ensemble couvrant.
- soit c^* la taille minimale d'un ensemble couvrant
- Ainsi, en posant $|Y| = 2p$ on a $p \leq c^*$, ce qui donne:

$$\frac{2p - c^*}{2p} \leq \frac{2p - p}{2p} = \frac{1}{2}$$

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 $\sum_{f \in F} f \quad = \quad x$?

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

Réduction : A partir de COUVRANT-MIN.

PARTITION est $\mathcal{NP}$ complet

Données d'entrée : Un ensemble fini d'entiers S

Problème : Existe-t-il $S_1 \subset S$ et $S_2 \subset S$ disjoints tels que
$\sum_{f \in S_1} = \sum_{f \in S_2}$ $S_1 \cup S_2 = S$?

Partition $\in \mathcal{NP}$ **:** clair.

Réduction : A partir de SUBSET-SUM.

RANGEMENT OPTIMAL

- $p_1, p_2, \ldots, p_n$ sont les poids des objets
- On souhaite les ranger dans un nombre minimum de boîtes, chaque boîte peut contenir un poids maximal de P.

Exemple

$$6.1,\ 4.1,\ 4,\ 4,\ 2,\ 1.9,\ 1.9 \quad P = 12$$

On peut faire en 2 boîtes

$$6.1 + 4 + 1.9 \qquad 4.1 + 4 + 2 + 1.9$$

Rangement glouton ou FIRST FIT DECREASING

Réordonner les objets par poids décroissants, ranger un à un les objets chacun dans la première boîte qui peut le contenir:

$$p_1 \geq p_2 \geq \ldots \geq p_n$$

- $s = 0$
- Pour $i = 1, \ldots n$ faire $\quad j = 1;$

  Tant que B_j ne peut pas contenir p_i faire $j++$

  Ajouter p_i à B_j

Sur l'exemple précédent $\quad 6.1,\ 4.1,\ 4,\ 4,\ 2,\ 1.9,\ 1.9 \quad P = 12$

On fait en 3 boîtes: $6.1 + 4.1; \qquad 4 + 4 + 2 + 1.9; \qquad 1.9$

Mais c'est le pire qui puisse arriver!

Théorème Rangement glouton donne une solution telle que le nombre de boîtes est inférieur à $1 + 3Nbopt/2$

Preuve:

Si tous les objets ont un poids $> \frac{P}{3}$ alors glouton donne l'optimum

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

Si par glouton chaque boîte contient un objet de poids $> \frac{P}{3}$ alors glouton donne l'optimum

Preuve : supprimer les objets de poids $\leq P/3$

S'il existe une boîte ne contenant que des objets de poids $\leq P/3$ alors toutes les boîtes sauf peut être une sont remplies à plus des 2/3

Soit b_j le poids contenu par la boîte j dans ce cas

$$\forall j < k \quad b_j > \frac{2P}{3}$$

$$\sum_{j=1}^{k} b_j > \frac{2P}{3}(k-1) + b_k$$

$$k_{opt} P > \frac{2P}{3}(k-1) + b_k$$

$$\frac{3k_{opt}}{2} > k - 1$$

Théorème S'il existe un algorithme polynomial $\varepsilon < 1/3$ approché pour Rangement optimal alors il existe un algorithme polynomial pour Partition et $P = NP$

Preuve:

- $E = \{e_1, e_2, \ldots, e_n\}$ instance de Partition
- $\mathcal{A}$ un algorithme ε-approché pour Rangement optimal
- On applique $\mathcal{A}$ à la suite E avec $P = \frac{\sum e_i}{2}$
- Si $\mathcal{A}$ répond 2 on a une solution à Partition
- Si $\mathcal{A}$ répond 3 ou plus on peut dire qu'il n'y a pas de solution à Partition car:

$$\frac{3-2}{3} > \varepsilon$$

Approximation efficace du sac à dos

On cherche les $x_i \in \{0, 1\}$ tels que :

$$\begin{cases} \sum_{i=1}^{n} c_i x_i \leq p \\ \max \sum_{i=1}^{n} a_i x_i \end{cases}$$

Proposition Pour tout $\varepsilon > 0$ il existe un algorithme polynomial ε-approché du problème du sac à dos.

On utilise la formulation de programmation dynamique :

$$P(i+1, v) = \min\{P(i, v), P(i, v - a_{i+1}) + p_{i+1}\}$$

où $P(i, v)$ est le poids minimal permettant de réaliser un bénéfice v avec les objets $1, 2, \ldots, i$.

- Posons $b_i = \lfloor \frac{a_i}{10^k} \rfloor$, k à déterminer
- La résolution du problème obtenu en remplaçant les a_i par les b_i donne une solution $\{x'_i\}$ en temps $O(\frac{n^2 M}{10^k})$
- Considérons cette solution optimale pour les b_i comme une solution approchée du problème initial et examinons l'écart vis à vis de la solution optimale $\{x_i\}$.

- De par la définition des b_i on a :

$$\sum_{i=1}^{n} a_i x'_i \geq \sum_{i=1}^{n} 10^k b_i x'_i$$

- Or x'_i est optimale pour le problème avec les b_i :

$$\sum_{i=1}^{n} b_i x'_i \geq \sum_{i=1}^{n} b_i x_i$$

- Mais on a $10^k(b_i + 1) > a_i$ ainsi:

$$\sum_{i=1}^{n} a_i x'_i \geq \sum_{i=1}^{n} (a_i x_i - 10^k) = (\sum_{i=1}^{n} a_i x_i) - n10^k$$

$$k \geq \log_{10} \frac{v\varepsilon}{n}$$

- un algorithme ε-approché qui fonctionne en temps $O(\frac{n^3}{\varepsilon})$

Approximation du voyageur de commerce

Proposition S'il existe $\varepsilon < 1$ et un algorithme polynomial ε-approché pour le problème du voyageur de commerce, alors $P = NP$.

Preuve:

- $\mathcal{A}$ un algorithme ε-approché pour Voyageur de commerce
- On montre que l'on peut résoudre le problème du cycle hamiltonien

- Soit $G = (X, E)$ un graphe pour lequel on cherche un cycle hamiltonien
- Soit k un entier supérieur à $\frac{1}{1-\varepsilon}$
- On value les arêtes par $v(x, y) = 1$ si $(x, y) \in E$
- $v(x, y) = nk + 1$ si $(x, y) \notin E$
- Si $\mathcal{A}$ répond n alors G admet un cycle hamiltonien
- Si $\mathcal{A}$ répond $x \neq n$ alors G n'admet pas de cycle hamiltonien

$$x \geq (n-1) + (nk+1)$$

$$\frac{n + nk - C_{opt}}{n + nk} \leq \varepsilon$$

$$Copt \quad \geq \quad n(1+k)(1-\varepsilon) \quad > \quad n$$

Backtracking, exploration arborescente

- Un problème de décision à résoudre (exemple SAT)
- On construit à chaque étape un ensemble S de problèmes; à la première étape S contient un seul problème: celui à resoudre
- A chaque étape, on choisit (astucieusement!) un des problèmes P que l'on supprime de S
- On décompose ce problème en une union de plusieurs problèmes $P_1, P_2, \ldots, P_k$ formant une liste L, telle que si l'un des P_i admet une solution alors on en déduit une solution pour P.
- On a ainsi:

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

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
- 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 s'il y en a. Sinon prendre une variable qui apparaît le plus grand nombre de fois dans une clause à 3 variables lui donner les valeurs vrai et faux
- Chaque problème se divise en 2, certains pouvant être simplement satisfaits ou par contre trivialement insatisfaisables.

Branch and Bound (B&B) ou séparation-évaluation (SEP)

- On a un problème d'optimisation
- On connnait une valeur satisfaisant aux contraintes
- On construit un arbre d'exploration de toutes les solutions satisfaisant les 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
- Pour cela il est nécessaire de connaître une fonction qui donne une borne pour la meilleure solution possible d'un sous arbre
- Exemple de la partition équitable d'un graphe

Un exemple : la partition équitable d'un graphe

- Un graphe $G = (X, E)$
- Chaque arête x, y a une valuation $v(x, y)$
- Il s'agit de diviser X en deux sous-ensembles X_1 et X_2 de cardinaux n_1, n_2 tels que :

$$|n_1 - n_2| \leq 1, \quad \min \sum_{x \in X_1,\ y \in X_2} v(x, y)$$

- Applications:
  - Décompositions d'un problème en sous-problèmes
  - Conception de circuits
  - Représentations graphiques

Mise en équations de la partition équitable d'un graphe

- Une variable $x_1, x_2, \ldots, x_n$ par sommet de X
- Cette variable prend la valeur 0 ou 1
- Les contraintes :

$$\frac{n}{2} - 1 \leq \sum_{i=1,n} x_i < \frac{n}{2} + 1$$

- Fonction à optimiser

$$\min \sum_{\{x_i,x_j\} \in E} v(x_i, x_j)|x_i - x_j|$$

Ingrédients de la méthode B&B

- Une méthode heuristique qui donne une solution initiale
- Un moyen de diviser un problème $\mathcal{P}$ en sous-problèmes $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k$ ayant chacun des contraintes supplémentaires et tels que :

$$Sol(\mathcal{P}) = \bigcup_{i=1,k} Sol(\mathcal{P}_i)$$

- Une fonction $g(\mathcal{P}_i)$ qui pour chaque problème $\mathcal{P}_i$ donne une *bonne* borne inférieure du coût optimum de $\mathcal{P}_i$
- Une stratégie de parcours de l'arbre des solutions

Algorithme de la méthode B&B

```
Calculer une solution heuristique H et soit C son cout;
     meilleur = C; solution = H;
AExplorer = {P}
Tant que (AExplorer != vide) faire
     Q = Tete(AExplorer);
     AExplorer = AExplorer - Q;
     Diviser Q en Q1 , Q2, ... ,Qk;
     Pour i = 1 a k faire
          Si Qi est admet une solution {
               calculer son cout Ci;
               Si (Ci < meilleur) {
                  meilleur = Ci; solution = Qi;}
                }
          Sinon{ calculer g(Qi);
               Si(g(Qi) < meilleur) ajouter(Qi, AExplorer);
               }
```

Ingrédients de la méthode B&B : la méthode heuristique

- Tout l'art de l'heuristique:
  - Un tirage aléatoire
  - Recuit simulé, recherche tabou, algo génétique etc.
  - Chaque cas requiert du savoir faire
- Exemple de la partition équitable d'un graphe

  On met x_1 dans X_1, puis on effectue l'algorithme suivant :

```
Pour i = 2 a n/2 faire
     Choisir le sommet y dont la somme des distances
          aux elements de X1 est  maximale
     Ajouter y a X1;
```

Ingrédients de la méthode B&B : la division d'un problème

- Il faut fixer une ou plusieurs nouvelles contraintes:
- En général on donne une valeur à une variable
- On examine toutes les valeurs possibles
- Choix astucieux de la variable en question
- Exemple de la partition équitable d'un graphe

  On prend une des variables x_j et on divise le problème en deux :

$$x_j \in X_1, \quad x_j \in X_2$$

Ingrédients de la méthode B&B : la fonction $g(\mathcal{P}_i)$

- C'est la grande question!!:
- Impossible de donner une technique générale, chaque cas requiert du savoir faire

- Exemple de la partition équitable d'un graphe

  Etant donnés deux sous-ensembles en cours de construction X_1 de cardinal $p < n/2$, X_2 de cardinal $q < n/2$, et un troisème ensemble Y de sommets non-encore affectés la meilleure solution aura un coût minimal égal à la somme des coûts des arêtes suivantes :

  1. celles entre X_1 et X_2.
  2. celles des $n/2 - q$ de moindre coût entre chaque élément de X_1 et Y.
  3. celles des $n/2 - p$ de moindre coût entre chaque élément de X_2 et Y.
  4. celles des $(n/2 - p)(n/2 - q)$ de moindre coût entre les éléments de Y.

Ingrédients de la méthode B&B : la stratégie de parcours

- Il faut se fixer un algorithme de parcours de l'arbre:
- Parcours en profondeur d'abord
- Parcours en largeur
- Choix astucieux qui permet de trouver une solution proche de l'optimum

Un exemple complet

	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	2	20	3	4	1
x_2	2	0	9	7	7	2
x_3	20	9	0	7	6	8
x_4	3	7	7	0	3	3
x_5	4	7	6	3	0	2
x_6	1	2	8	3	2	0

La méthode (B&B) pour la programmation linéaire en nombres entiers

- On a un problème du type

$$\begin{cases} Ax \leq b \\ \max cx \end{cases}$$

- Avec la contrainte supplémentaire que la solution doit être un entier
- On construit un ensemble de problème linéaires en nombres rationnels à résoudre
- A chaque étape on doit résoudre tous ceux qui sont en cours, si tous ont une solution entière alors on prend celle qui rend maximum la fonction économique
- Si certaines solutions sont rationnelles (par exemple on trouve : $x_i = p/q$) alors on remplace le problème en question par deux nouveaux problèmes l'un contenant la contrainte supplémentaire $x_i \leq \lfloor p/q \rfloor$ l'autre la contrainte supplémentaire $x_i \geq \lceil p/q \rceil$
- On montre que l'algorithme termine toujours

Algorithme de la méthode B&B pour la programmation en nombre entiers

```
Initialiser les systemes a resoudre par
     S = {Ax <= b , max cx}
Tant que (S != vide) faire {
     Resoudre tous les systemes dans S
     Si toutes les solutions sont entieres{
          choisir celle qui rend maximum cx;
           exit;}
     Supprimer de S les P qui n'ont pas de solution;
     Soit  pour un certain P dans S un  xj de la solution
          qui obtient une valeur non-entiere
         supprimer P de S et ajouter deux systemes a S,
         l'un contenant les contraintes de P et en plus xj <= a
         l'autre contenant les contraintes de P et xj >= a+1

   }
```