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 : $6.1 + 4.1$; $4 + 4 + 2 + 1.9$; 1.8

Théorème FIRST FIT DECREASING utilise $1 + 3\ \mathrm{OPT}/2$ 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

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

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

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:

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

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 $f = \sum_i f_i$.

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

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 connnait 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 $x_i \geq \lfloor \alpha_i \rfloor + 1$