B3 – SET-COVER et autres problèmes NP-complets

ou Comment en faire le moins possible à l'agrégation

Voici un sujet de TP illustrant le caractère NP-complet du problème de couverture minimale (SET-COVER) en comparant une approche de résolution gloutonne et une approche optimale à l'aide du langage Python et du package de programmation linéaire `glpk`.

1 Préliminaires

Le problème d'optimisation SET-COVER-OPT. Étant donné un ensemble L et des parties D_1, ..., D_n recouvrant L, on s'intéresse à déterminer un ensemble $I \subseteq \{1, \ldots, n\}$ de cardinal minimal tel que $\bigcup_{i \in I} D_i = L$.

Application à la préparation de l'agrégation. Quel est le nombre minimal de développements à apprendre pour en garantir au moins un par leçon ?

Le problème de décision associé SET-COVER. Étant donné un ensemble L, des parties D_1, ..., D_n recouvrant L $\boxed{\text{et un entier } k \geqslant 1}$, on s'intéresse à savoir s'il existe un ensemble $I \subseteq \{1, \ldots, n\}$ tel que $\boxed{|I| \leqslant k}$ et $\bigcup_{i \in I} D_i = L$.

3D-PERFECT-MATCHING. Soit A, B, C trois ensembles finis de même cardinal n et T une partie de $A \times B \times C$. Une partie M de T est dite *couplage triparti* si deux éléments quelconques de M n'ont aucune composante en commun, c'est-à-dire si pour toute paire $\{(a_1, b_1, c_1), (a_2, b_2, c_2)\}$ d'éléments de M, on a : $a_1 \neq a_2$, $b_1 \neq b_2$ et $c_1 \neq c_2$. Étant donné un tel ensemble T, on s'intéresse à savoir s'il admet un couplage triparti de cardinal n.

INTEGER-LINEAR-PROGRAMMING. Un programme linéaire entier sous forme canonique s'exprime de la manière suivante :

$$\begin{array}{lll} \text{Minimiser} & c^T x & \\ \text{Soumis à} & Ax \leqslant b & \text{pour } \begin{cases} A \text{ une matrice à coefficients entiers} \\ b \text{ et } c \text{ des vecteurs à coefficients entiers.} \end{cases} \\ & x \geqslant 0 & \\ & x \in \mathbb{Z}. & \end{array}$$

2 Énoncé

Exercice 1. Prouver que SET-COVER est NP-complet, en prouvant qu'il est dans NP et en faisant une réduction depuis 3D-PERFECT-MATCHING.

Exercice 2. Montrer que INTEGER-LINEAR-PROGRAMMING est NP-complet en prouvant qu'il est dans NP et en exprimant une instance de SET-COVER sous la forme d'une instance de INTEGER-LINEAR-PROGRAMMING.

Exercice 3. Implémenter l'algorithme glouton qui à partir d'une instance $\langle L, D\rangle$ de SET-COVER-OPT construit une partie I en choisissant à chaque étape l'ensemble contenant le plus d'éléments non encore couverts.

Exercice 4. Écrire un programme Python qui prend en entrée une instance de SET-COVER et renvoie une instance de INTEGER-LINEAR-PROGRAMMING sous le format CPLEX LP [1]. Vérifier avec des exemples d'instances que l'approche gloutonne n'est pas optimale.

Exercice 5. (Chvatal, 1979) a prouvé que l'algorithme glouton réalise une H_m-approximation de SET-COVER-OPT où m est la taille maximale d'un ensemble de D et H_n est le n-ième terme de la série harmonique. Trouver une instance de SET-COVER-OPT pour laquelle l'algorithme glouton réalise un ratio d'approximation de $\frac{1}{2}\log_2|L|$.

Exercice 6. Combien de développements suffisent à obtenir 2 développements par leçon pour l'option D, à partir des données sur `http://agreg-cachan.fr` ?

3 Corrigé

Exercice 1. SET-COVER est dans NP car la donnée de I permet de vérifier en temps polynomial que $\bigcup_{i\in I} D_i = L$. Étant donné une instance $I_{3D} = \langle A, B, C, T\rangle$, quitte à renuméroter les éléments de A, B et C on peut les supposer disjoints. On crée alors l'instance I_S de SET-COVER où l'ensemble à recouvrir est $A \cup B \cup C$, les parties D_i (pour $i \in \{1, \ldots, |T|\}$) correspondent aux triplets de T vus comme ensembles à 3 éléments et k vaut $|A|$. Ainsi, T admet un couplage triparti de cardinal $|A|$ si et seulement si on peut recouvrir l'ensemble $A \cup B \cup C$ de cardinal $|A|+|B|+|C| = 3|A|$ avec au plus $|A|$ parties à 3 éléments associées aux éléments de T, ces $|A|$ parties étant forcément disjointes par cardinalité. Donc SET-COVER est NP-complet. Une preuve que 3D-PERFECT-MATCHING est NP-complet se trouve dans (Dasgupta et al., 2006).

1. Voir `http://lpsolve.sourceforge.net/5.0/CPLEX-format.htm`.

Exercice 2. INTEGER-LINEAR-PROGRAMMING est dans NP car étant donné une solution, on peut vérifier les contraintes en temps polynomial. Étant donné une instance $\langle L, (D_1, \ldots, D_n)\rangle$ de SET-COVER, on va encoder les parties I possibles de $\{1, \ldots, n\}$ par un vecteur $x \in \{0,1\}^n$ tel que : $x_i = 1$ si et seulement si $i \in I$. Ainsi, si on numérote les éléments de $L = \{\ell_1, \ldots, \ell_m\}$ et si on définit la matrice A où a_{ij} vaut 1 si $\ell_i \in D_j$ et 0 sinon, on a pour tout x correspondant à un choix de I et pour tout $i \in \{1, \ldots, m\}$, la i-ième composante de Ax vaut le nombre de D_j contenant ℓ_i lorsque j décrit I. Le problème de INTEGER-LINEAR-PROGRAMMING est alors exactement :

$$\begin{array}{ll} \text{Minimiser} & |I| = 1^T x \\ \text{Soumis à} & Ax \geqslant 1 \\ & x \geqslant 0 \\ & x \leqslant 1 \\ & x \in \mathbb{Z}. \end{array}$$

Exercice 3. Voici une implémentation possible en Python.

```
def glouton(L, D):
    m = len(L)
    n = len(D)
    couverts = set()
    choix_glouton = []
    while len(couverts) < m:
        nb_couverts_max = 0
        i_max = None
        for i in range(n):
            nb_couverts = len(set(D[i]) - couverts)
            if nb_couverts > nb_couverts_max:
                nb_couverts_max = nb_couverts
                i_max = i
        choix_glouton.append(i_max)
        couverts.update(D[i_max])
    return choix_glouton
```

Exercice 4. On construit d'abord la matrice A de l'instance correspondante de INTEGER-LINEAR-PROGRAMMING.

```
def export_cplexlp(L, D):
    A = [[1 if element in partie else 0 for partie in D] for element in L]
    nb_variables = len(D)
    print 'Minimize'
    print '   obj: ' + ' + '.join('x%d' % i for i in range(nb_variables))
    print 'Subject To'
```

```
    for line in A:
        l = ['x%d' % i for i, v in enumerate(line) if v == 1]
        if l:
            print '   ' + ' + '.join(l) + ' >= 1'
    print 'Binary'
    print '   ' + ' '.join('x%d' % i for i in range(nb_variables))
```

Exercice 5. On considère un ensemble L à $2^{k+1} - 2$ éléments. On définit $k + 2$ parties comme suit : les parties $D_1, \ldots, D_k$ sont choisies disjointes et telles que D_i a 2^i éléments (elles recouvrent donc L) et on y ajoute D_{k+1} et D_{k+2} telles que pour chaque D_i pour i de 1 à k, la moitié de ses éléments sont dans D_{k+1}, l'autre moitié sont dans D_{k+2} (ces deux dernières recouvrent donc L également). L'algorithme glouton renvoie $I = \{k, \ldots, 1\}$ tandis qu'une solution optimale est réalisée par $I = \{k + 1, k + 2\}$ ce qui donne un ratio d'approximation de $k/2 \leqslant \frac{1}{2} \log_2 |L|$.

```
def instance_gourmande(k):
    L = range((1 << k + 1) - 2)
    D = [L[(1 << i) - 2:(1 << i + 1) - 2] for i in range(1, k + 1)]
    t0 = []
    t1 = []
    for partie in D:
        t0.extend(partie[:len(partie) / 2])
        t1.extend(partie[len(partie) / 2:])
    D.append(t0)
    D.append(t1)
    return L, D
```

Exercice 6. La réponse est 42.

Références

Vasek Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3) :233–235, 1979.

Sanjoy Dasgupta, Christos H Papadimitriou, and Umesh Vazirani. *Algorithms*. McGraw-Hill, Inc., 2006.