

1. Cours

cf Kleinberg-Tardos, sec 12.6

2. Équilibrage de charge

i.a. Il suffit de montrer qu'on ne peut mener une suite infinie d'échanges favorables. Comme une telle suite boucle nécessairement, considérons une suite d'échanges qui ramène à la situation de départ et restreignons nous à l'ensemble des machines qui effectuent un échange dans cette suite. Considérons l'évolution de la charge maximale sur ces machines : celle-ci est nécessairement constante, ce qui est contradictoire avec le fait que la machine qui la porte effectue un échange favorable.

i.b. Montrons que la charge maximale est au plus $2 \max(\text{moyenne des charges}, \max p_i)$. Remarquons que pour toute affectation, et donc en particulier pour l'optimum, la charge maximale est au moins $\max(\text{moyenne des charges}, \max p_i)$.

Considérons la machine la moins chargée de notre affectation : sa charge M_- est au plus $\frac{1}{m} \sum_i p_i$, et la machine la plus chargée de notre affectation, dont la charge est M_+ .

Il n'y a pas d'échanges favorables possibles entre ces deux machines, donc $M_+ \leq \frac{1}{2}(M_+ + M_- + \max p_i)$ puisque sur deux machines on peut toujours équilibrer de manière gloutonne et placer la plus grande tâche en dernier sur la machine la moins chargée. Ceci donne

$$M_+ \leq M_- + \max p_i \leq 2 \max\left(\frac{1}{m} \sum_i p_i, \max p_i\right) \leq 2\text{OPT}$$

ii. Dans la version online considérons la machine j qui porte la charge maximale. La dernière tâche k qui a été ajoutée sur j l'a été à un moment où cette machine était la moins chargée. La charge de j était alors inférieure à la moyenne des charges des tâches déjà arrivées :

$$\text{Online} \leq \frac{1}{m} \sum_{i < k} p_i + p_k \leq 2 \max\left(\frac{1}{m} \sum_i p_i, \max p_i\right).$$

Pour être plus précis on revient aux deux inégalités $\text{OPT} \geq \max p_i$ et $\text{OPT} \geq \frac{1}{m} \sum p_i$ et on écrit

$$\text{Online} \leq \frac{1}{m} \sum_{i < k} p_i + p_k \leq \frac{1}{m} \sum_{i \leq k} p_i + \left(1 - \frac{1}{m}\right)p_k \leq \left(2 - \frac{1}{m}\right)\text{OPT}.$$

3. Arbre d'évolution optimal pour la distance de Hamming.

i. On considère le graphe complet à m sommets où chacun des sommets représente un des f_i et où l'arête entre f_i et f_j a un poids égal à la distance de Hamming entre ces deux mots.

On peut ensuite appliquer l'algorithme de Kruskal sur ce graphe pour obtenir un arbre d'évolution de coût minimal.

ii. On utilise la construction donnée par l'énoncé. Étant donné un arbre T d'évolution de coût k , on construit un ensemble couvrant de sommets. Remarquons que toutes les arêtes sont de poids

au plus 2, en effet w_\emptyset appartient à l'arbre et est à distance au plus 2 de tous les éléments. S'il existe une arête entre deux mots qui sont à distance égale à deux alors on peut diviser cette arête en une chaîne de 2 arêtes ayant chacune poids 1 et telle que tous les sommets sur cette chaîne appartiennent à F ou à G .

Soit T' l'arbre ainsi obtenu (dont le coût est le même que T). On peut à nouveau modifier T' sans changer son coût pour qu'il soit enraciné en w_\emptyset et soit de hauteur 2. Dans T' , chaque sommet f_p de $F \setminus \{w_\emptyset\}$ a un voisin dans G qui est par construction une extrémité de f_p dans le graphe Γ . En sélectionnant pour chaque sommet f_p de F un de ses voisins, on obtient un ensemble de sommets de G couvrant toutes les arêtes et de cardinal inférieur à $k - m$.

Réciproquement, étant donné un ensemble couvrant V de sommets de cardinal inférieur ou égal à k . On construit un graphe d'évolution enraciné en w_\emptyset et dont les fils sont l'ensemble V . On ajoute enfin les m feuilles de l'arbre, ce qui donne un arbre d'évolution de poids égal à $k + m$.

4. Placement optimal de ressources

i. La version de décision du problème consiste à se fixer un entier k et à répondre à la question : existe-t-il un ensemble dominant de taille inférieure ou égale à k ?

ii. Soit $(E, (S_i)_{i \in I})$ une instance de recouvrement par ensemble où les S_i sont des sous-ensembles de E . On construit le graphe dont l'ensemble des sommets est $V = E \cup I$ tel que sa restriction à I soit le graphe complet et il existe une arête $(e, i) \in E \times I$ si et seulement si $e \in S_i$.

Soit X un ensemble dominant du graphe de taille k . On peut supposer que tous les sommets de X appartiennent à I , en effet si $x \in X$ appartient à E alors peut-être remplacé par n'importe lequel de ses voisins.

Cela nous donne donc une solution au problème de recouvrement d'ensembles de taille inférieure ou égale à k .

iii. Soit S la solution au problème k -CENTRES. Si $\phi(S) = 1$, alors S est un ensemble dominant du graphe G , puisque pour tout $y \notin S$, il existe $x \in S$ tel que $d(x, y) = 1$ ce qui signifie que (x, y) est une arête de G .

iv. S'il existe un algorithme polynomial qui renvoie un ensemble S de taille k tel que $\phi(S) < 2\phi^*$, alors dans la construction précédente, on obtiendrait une solution au problème. Il est en effet évident que $\Phi(S) = 1$ ou 2. On aurait donc un algorithme polynomial pour résoudre ensemble dominant.

5. Programmation dynamique : Alignement de séquences

i. Définissons $s(i, j, k)$ comme étant le meilleur alignement entre $u_1, \dots, u_i, v_1, \dots, v_j$ et w_1, \dots, w_k . On établit alors facilement la récurrence, de façon analogue à ce qui se passe dans le cours pour le cas de deux chaînes :

$$s(i, j, k) = \min \begin{cases} s(i-1, j-1, k-1) + \Delta(u_i, v_j, w_k) \\ s(i-1, j-1, k) + \Delta(u_i, v_j, -) \\ s(i-1, j, k-1) + \Delta(u_i, -, w_k) \\ s(i, j-1, k-1) + \Delta(-, v_j, w_k) \\ s(i-1, j, k) + \Delta(u_i, -, -) \\ s(i, j-1, k) + \Delta(-, v_j, -) \\ s(i, j, k-1) + \Delta(-, -, w_k). \end{cases}$$

On a posé $\Delta(\alpha, \beta, \gamma) = \delta(\alpha, \beta) + \delta(\alpha, \gamma) + \delta(\beta, \gamma)$.

L'initialisation se fait par $s(0, 0, 0) = 0$ et $s(i, j, k) = +\infty$ si i, j ou k est strictement négatif.

La complexité est le coût du remplissage de la table, soit $O(n^3)$ cases, le calcul coutant $O(1)$ par case, soit $O(n^3)$ en tout.

ii. L'intuition naturelle consiste à poser $w(i', i, j', j)$ comme le meilleur alignement de $u_{i'} \dots u_i$ avec $v_{j'} \dots v_j$, puis à chercher à calculer $w(i', i, j, j')$; cela conduit cependant au mieux à un algorithme en $O(n^4)$.

On définit alors $t(i, j)$ comme le meilleur alignement d'un suffixe de $u_1 \dots u_i$ avec un suffixe de $v_1 \dots v_j$, ie : $t(i, j) = \max_{i' \leq i, j' \leq j} w(i', i, j', j)$.

La relation de récurrence pour $t(i, j)$ est similaire à celle obtenue dans le cas d'alignement de deux chaînes; la seule différence est le fait que le meilleur alignement d'un suffixe peut correspondre au cas où l'un des deux suffixes est trivial (et dans ce cas l'autre l'est nécessairement). Cela conduit à :

$$t(i, j) = \max \begin{cases} t(i-1, j-1) + \delta(u_i, v_j) \\ t(i-1, j) + \delta(u_i, -) \\ t(i, j-1) + \delta(-, v_j) \\ 0. \end{cases}$$

Une fois la table construite, le maximum permet d'obtenir les positions i et j où se terminent α et β , ainsi que la valeur de l'alignement optimal. Pour déterminer l'endroit où commencent α et β , il suffit de chercher à construire l'alignement optimum en remontant dans la table de programmation dynamique. Le premier 0 trouvé marque le point où les chaînes α et β commencent.

6. NP-complétude de ORD-TACHES

i) Soit $e = \{u, v\} \in E$. La décomposition est une partition de $X \cup E$, donc u figure dans l'un des ensembles de la décomposition, disons S_i , ainsi que e , disons dans S_j . On a $(u, e) \in A$, donc nécessairement par définition $1 \leq i < j \leq 3$. En particulier, $j \neq 1$ et $i \neq 3$, ce qui signifie $S_1 \cap E = S_3 \cap X = \emptyset$.

La même inégalité montre que $j = 2 \Rightarrow i = 1$, donc $S_2 \cap E$ est un sous-ensemble de $\{\{u, v\}, u \in S_1, v \in S_1\}$; cet ensemble est de cardinal $k(k-1)/2$, d'où $|S_2 \cap E| \leq k(k-1)/2$. Comme de fait $S_2 \cap E = \{\{u, v\}, u \in S_1, v \in S_1\} \cap E$, on voit que l'égalité n'a lieu que si $\{\{u, v\}, u \in S_1, v \in S_1\} \subset E$, ce qui signifie que toutes les arêtes entre éléments de $S_1 \cap X$ sont dans E , ou encore que $S_1 \cap X$ est une clique de G .

ii) On suppose Y, Z et T disjoints; à partir de là, le calcul est simple : le nombre d'arcs est $|A| + pq + qr$. Or A contient deux éléments par arête de G , donc $|A| = 2m$.

Soit C une clique de taille k de G . On peut alors construire la décomposition compatible de Γ : $S_1 = C$, $S_2 = (C \times C) \cup (X - C)$, $S_3 = E - (C \times C)$; on peut en déduire une décomposition compatible de Γ' sous la forme $S_1 \cup Y$, $S_2 \cup Z$, $S_3 \cup T$. Dans ce cas, les cardinaux des ensembles sont $k+p, k(k-1)/2+n-k+q, m-k(k-1)/2+r$, et pour tout $l \geq \max(k, k(k-1)/2+n-k, m-k(k-1)/2)$ on peut choisir p, q, r de sorte que la décomposition compatible de Γ' soit en trois sous-ensembles de même taille l .

Inversement, supposons que S admette une décomposition compatible en trois sous-ensembles de taille l . Un argument analogue à celui de la question i) montre que nécessairement $Y \subset S_1$, $Z \subset S_2$ et $T \subset S_3$. Par suite, $|S_1 \cap X| = l - p = k$, et $|S_2 \cap X| + |S_2 \cap E| = l - q = k(k-1)/2 + n - k$, d'où, vu $S_3 \cap X = \emptyset$, $|S_2 \cap E| = k(k-1)/2$ et $S_1 \cap X$ est une clique de taille k de G .

iii) ORD-TACHES est manifestement dans \mathcal{NP} : la donnée d'une décomposition compatible, qui est de taille polynomiale, fournit un certificat vérifiable en temps polynomial, puisqu'il suffit, pour chaque i et chaque sommet de S_i d'énumérer les arêtes (x, y) issues de x et de vérifier que y est dans un ensemble S_j avec $j > i$; le coût est, au pire, un parcours de A et un parcours de S pour chaque sommet, donc $O(n^2m)$. La question ii) montre que CLIQUE se réduit à un cas particulier de ORD-TACHES, celui où l'on fixe $t = 3$ et $l = |S|/3$; cette instance, et donc le problème ORD-TACHES plus généralement, est donc bien \mathcal{NP} -complet.

Remarque : l'ajout des ensembles Y, Z, T peut paraître artificiel, mais il permet de se ramener à l'énoncé initial d'ORD-TACHES. La construction de Γ permettait de montrer qu'il existait une clique de taille k si et seulement s'il existait une décomposition compatible avec $|S_1| = k$ et $|S_2| = k(k-1)/2 + n - k$, ce qui ne se réduit pas de façon aisée à l'existence d'une décomposition compatible dont tous les ensembles sont de taille $\leq l$. Ajouter Y, Z et T a fourni le degré de liberté nécessaire.

7. NP-complétude de MAX2SAT

i) La façon la plus simple d'énumérer les différentes possibilités consiste à remarquer que la formule est symétrique en x, y, z . On peut donc se contenter de distinguer selon le nombre de valeurs 1 (= vrai) parmi ces trois littéraux.

nb. 1	t	nb sat.
0	0	6
0	1	4
1	0	7
1	1	5
2	0	7
2	1	5
3	0	6
3	1	7

Il s'ensuit en particulier que le nombre de clauses satisfaites est au plus 7, et qu'il existe une valeur de t telle que ce nombre soit 7 si et seulement si la clause $x \vee y \vee z$ est satisfaite.

ii) On va réduire polynomialement 3SAT à MAX2SAT. Soit une formule de la forme $f_1 \wedge f_2 \wedge \dots \wedge f_n$, où chaque f_n dépend des variables x_1, \dots, x_m et est de la forme $x_{i_1} \vee x_{i_2} \vee x_{i_3}$.

On crée alors des variables t_i , et l'on remplace f_i par g_i donnée par

$$x_{i_1}, x_{i_2}, x_{i_3}, t_i, (\overline{x_{i_1}} \vee \overline{x_{i_2}}), (\overline{x_{i_2}} \vee \overline{x_{i_3}}), (\overline{x_{i_3}} \vee \overline{x_{i_1}}), (x_{i_1} \vee \overline{t_i}), (x_{i_2} \vee \overline{t_i}), (\overline{x_{i_3}} \vee \overline{t_i}).$$

Il est clair à partir de la question 1 que si la formule initiale est satisfiable, il existe un choix des valeurs des t_i tel qu'exactly 7n des clauses de g_1, \dots, g_n soient satisfaites. À l'inverse, si l'on dispose d'un choix de valeurs des variables x_k et t_i tel que 7n clauses soient satisfaites, la question 1 montre que chacune des clauses f_i est satisfaite par cette affectation de variables. Résoudre 3SAT sur la formule initiale équivaut donc à résoudre MAX2SAT sur le problème construit.

Qui plus est, le nombre de variables et de clauses dans le problème ainsi construit est linéaire, donc polynomial, en le nombre de clauses et de variables initial. On a donc bien construit une réduction polynomiale de 3SAT à MAX2SAT.

Comme de plus MAX2SAT est dans \mathcal{NP} (la donnée d'une affectation de valeurs aux variables fournit un certificat vérifiable en temps linéaire), il s'ensuit que MAX2SAT est \mathcal{NP} -complet.