École polytechnique | Majeure informatique
Promotion 2004

Cours « Conception et analyse d'algorithmes »
PC n^o 4 – mariages stables et programmation linéaire

3. Plus court chemin.

Soit $G = (V, E)$. On associe à chaque sommet v une variable $d(v)$. On considère le programme linéaire suivant :

$$\max(d(t) - d(s))$$

sous les contraintes : $\forall (u, v) \in E \quad d(v) - d(u) \leq c(u, v)$.

Considérons un chemin $\gamma = (x_i)_{1 \leq i \leq n}$ avec $x_1 = s$, $x_n = t$ et $(x_i, x_{i+1}) \in E$ pour tout $i < n$. Si $d_0(v)$ est une solution aux contraintes du problème linéaire ci-dessus, alors on a

$$d(t) - d(s) = \sum_{i=1}^{n-1} d(x_{i+1}) - d(x_i) \leq \sum_{i=1}^{n-1} c(x_i, x_{i+1}),$$

qui est exactement la longueur du chemin γ. Il s'ensuit que $d(t) - d(s)$ minore en particulier la longueur du plus court chemin de s à t dans G.

Inversement, si l'on prend comme valeurs des variables $d(v)$ les longueurs $\delta(v)$ des plus courts chemins de s à v, on a manifestement $d(v) - d(u) \leq c(u, v)$, sans quoi on obtient un chemin plus court pour aller en v en suivant un plus court chemin vers u puis en prenant l'arête (u, v) ; les valeurs $d(v) = \delta(v)$ vérifient donc le système de contraintes. Par conséquent, $\max(d(t)-d(s)) \geq \delta(t)-\delta(s) = \delta(t)$. Mais on a vu que $\max(d(t)-d(s)) \leq \delta(t)$ dans la première partie, d'où l'égalité (dans ces deux dernières inégalités, le maximum est bien entendu pris sur l'ensemble des solutions au système de contraintes).

Le dual du programme linéaire s'écrit, quant à lui, en associant à chaque arête une variable $f(u, v)$,

$$\min \sum_{(u,v) \in E} c(u, v) f(u, v)$$

sous les contraintes :

$$\begin{aligned}
\sum_{(v,u)\in E} f(u, v) - \sum_{(u,v)\in E} f(u, v) &= 0, \forall u \in V - \{s, t\} \\
\sum_{(v,s)\in E} f(v, s) - \sum_{(s,v)\in E} f(s, v) &= -1 \\
\sum_{(v,t)\in E} f(v, t) - \sum_{(t,v)\in E} f(t, v) &= 1 \\
f(u, v) &\geq 0.
\end{aligned}$$

Les contraintes sont des contraintes de type flot, hormis le fait qu'il peut exister un flot sortant de t ou entrant en s, et que toutes les arêtes sont de capacité infinie.

La première contrainte traduit la conservation du flot, les deux suivantes indiquant que le flot total sortant de s vaut 1 et le flot total entrant en t vaut 1. Il ne s'agit donc pas d'un problème de flot maximal (voir exercice 4), mais d'un problème ou la quantité de flot est fixée (ici unitaire), et où le flot est pondéré par des coefficients $c(u,v)$, et on veut que son coût soit minimal ; ce type de problème est appelé *min-cost flow*. L'exercice montre que ce problème se réduit, par dualité, à un problème de plus court chemin. Noter le parallèle avec Ford-Fulkerson, où l'augmentation du flot se réduisait à un problème d'atteignabilité.

4. Adéquation linéaire

Supposons que les mesures n'ont pas toutes la même valeur x_i, sans quoi la droite d'adéquation est simplement la droite verticale $x = x_i$. Dans le cas contraire, on introduit trois variables a, b, m, m étant le maximum des distances entre les points et la droite $y = ax + b$.

Cette hypothèse sur m se traduit par les contraintes $-ax_i - b + m \geq -y_i$ et $ax_i + b + m \geq y_i$. La fonctionnelle linéaire à minimiser est tout simplement m. La seule contrainte de signe est $m \geq 0$. On notera C_i la i-ème contrainte du premier type, C_{i+n} la i-ème contrainte du second type.

Notons que ce problème linéaire a peu de variables mais énormément de contraintes. Le dual, quant à lui, s'écrit

$$\begin{cases} \max \sum y_i(z_{i+n} - z_i), \\ \sum x_i(z_{i+n} - z_i) = 0, \\ \sum z_{i+n} - z_i = 0, \\ \sum z_{i+n} + z_i \leq 1. \end{cases}$$

Il est possible, quand le nombre de points est grand, que sa géométrie soit plus simple, et qu'il soit par conséquent plus facile à résoudre par la méthode du simplexe.

5. Ordonnancement préemptif sur des machines parallèles

i) Soit $C = \max\left(\max p_i, \frac{\sum p_i}{m}\right)$. C est naturellement une borne inférieure de la date de fin. Nous décrivons maintenant un algorithme qui renvoie un ordonnancement réalisable de longueur égale à C. A l'initialisation, la machine courante M_c est la première machine et la date courante t_c est la date origine. Pour chaque tâche i (l'ordre de sélection n'importe pas), si $f = t_c + p_i \leq C$, i est exécutée sur M_c entre les dates t_c et e. La nouvelle date courante devient $t_c \leftarrow e$. Si $f > C$, la tâche i est exécutée en deux parties : sur M_c entre t_c et C et sur $M_c + 1$ (qui devient machine courante entre 0 et $f - C$ (cette dernière date devenant date courante). puisque $C \geq p_i$, on a $f - C \leq t_c$ ce qui prouve que les deux parties de l'exécution de i sont bien disjointes dans le temps. On prouve facilement que la machine courante lorsqu'on a placé i est $\lceil \sum_{k=1}^{i} p_k / C \rceil$. La définition de C montre alors que le numéro de la machine courante ne dépasse pas m. On constate que chaque tâche est exécutée en une ou deux parties.

ii) On classe les dates r_i et d_i de manière croissante, c'est-à-dire qu'on détermine $\theta_1 \leq \theta_2 \leq \cdots \leq \theta_{2n}$ tels que chaque θ_k correspond à un r_j ou à un d_j. Soit x_{ik} $(1 \leq i \leq n$ et $1 \leq k < 2n)$ la longueur de la tâche i exécutée entre les dates θ_i et θ_{i+1}. Pour chaque tâche i, la contrainte que la durée totale de i est p_i s'écrit :

$$\sum_{1 \leq k < n} x_{ik}$$

Les contraintes de fenêtre de temps s'écrivent

$$x_{ik} = 0 \text{ si } \theta_{k+1} \leq r_i \text{ ou } \theta_k \geq d_i$$

Sur chaque intervalle, la durée maximale d'une tâche est limitée par la longueur de l'intervalle et la charge totale est limitée par la charge disponible :

$$x_{ik} \leq \theta_{k+1} - \theta_k \text{ pour tout } i \text{ (et tout } k)$$

$$\sum_{1 \leq i \leq n} x_{ik} \leq m(\theta_{k+1} - \theta_k)$$

Réciproquement si on trouve des x_{ik} satisfaisant toutes ces inégalités, il est possible, d'après la première question, de construire un ordonnancement réalisable. On utilise en effet l'algorithme décrit en i) sur chaque intervalle $[\theta_k, \theta_{k+1}]$.

On obtient un algorithme d'optimisation en remplaçant la contrainte fixant la valeur de θ_{2n} par l'inégalité $\theta_{2n-1} \leq \theta_{2n}$. θ_{2n} n'est plus alors une date fixe et représente la date de fin de l'ordonnancement. On minimise θ_{2n} sous les contraintes ci-dessus. Si ce programme linéaire a une solution, on fait la même chose avec θ_{2n-1} et ainsi de suite jusqu'à ce qu'on arrive a un programme sans solution. La date de fin minimale correspond à la dernière solution obtenue.