GIPAD – GS2
Recherche Opérationnelle - APP1
Correction devoir final

12 octobre 2009

durée : 3 heures

documents autorisés : aucun. La dernière section de ce document rappelle un nombre de définitions et de résultats utiles.

barème : note sur 20 = nombre de points * 20 / 30
- problème 1 : 6 points (modélisation PLNE)
- problème 2 : 8 points (modélisations graphes)
- problème 3 : 8 points (recherche locale)
- problème 4 : 8 points (dualité)
- points supplémentaires pour questions bonus (marquées *).

---

1 TSPTW : programmation linéaire en nombres entiers

Question 1 ***(6 points)***

Q1.1. définir le problème du voyageur de commerce euclidien asymétrique avec fenêtres de temps.

Q1.2. modéliser le problème en un programme linéaire en nombres entiers.

Q1.3. décrire chaque famille de contraintes et de variables du modèle.

Le TSPTW consiste à trouver un tour de coût total minimal sur un ensemble de villes tel que l'instant de passage en chaque ville est compatible avec une fenêtre de temps donnée. On considère un ensemble $N = \{1, \ldots, n\}$ de villes et une ville de départ et d'arrivée notée 0. On note A l'ensemble des arcs entre paires de villes. À chaque arc (i, j) est associé un coût c_{ij} et une durée $t_{ij} > 0$. La matrice des coûts vérifie l'inégalité triangulaire (TSP eulérien : $c_{ij} + c_{jk} \geqslant c_{ik}$). À chaque ville i est associée une fenêtre de temps $[a_i, b_i]$ contraignant l'instant d'arrivée dans la ville i en partant de la ville 0 à l'instant 0. Un tour est un cycle élémentaire (car graphe eulérien) hamiltonien dans le graphe orienté $G = (N \cup \{0\}, A)$ et son coût est la somme des coûts des arcs du cycle. L'instant d'arrivée du tour dans une ville i est au moins égal (un temps d'attente dans les villes intermédiaires est autorisé) à la somme des durée des arcs du cycle joignant le sommet 0 au sommet i. Pour chaque ville i, cet instant doit être compris entre

a_i et b_i.

$$\min\ z = \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1}$$

$$\text{s.t.} \sum_{j|(i,j)\in A} x_{ij} = 1 \qquad \forall i \in N\cup\{0\}, \tag{2}$$

$$\sum_{j|(j,i)\in A} x_{ji} = 1 \qquad \forall i \in N\cup\{0\}, \tag{3}$$

$$w_j - w_i \geqslant t_{ij} - M_{ij}(1-x_{ij}) \qquad \forall (i,j)\in A, i\neq 0, \tag{4}$$

$$w_j \geqslant t_{0j} - M_j(1-x_{0j}) \qquad \forall j)\in N, \tag{5}$$

$$a_i \leqslant w_i \leqslant b_i \qquad \forall i \in N\cup\{0\}, \tag{6}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j)\in A, \tag{7}$$

$$w_i \in \mathbb{R}_+ \qquad \forall i \in N\cup\{0\}. \tag{8}$$

On définit pour chaque arc (i,j) une variable binaire x_{ij} égale à 1 si et seulement si l'arc appartient au cycle, et pour chaque ville i une variable w_i modélisant l'instant d'arrivée dans la ville i. Les contraintes (2) et (3) forcent à passer par chaque ville exactement une fois. Les contraintes (6) modélisent les fenêtres de temps. Les contraintes (4) et (5) modélisent les précédences : si l'arc (i,j) (resp. l'arc $(0,j)$) est emprunté alors w_j est supérieur à $w_i + t_{ij}$ (resp. t_{0j}). Si l'arc (i,j) n'est pas emprunté $x_{ij} = 0$, la contrainte ne doit pas être active (elle doit être toujours vraie) ; pour cela, il suffit de poser $M_{ij} = t_{ij} - a_j + b_i$ (resp. $M_j = t_{0j} - a_j$). Enfin ces contraintes interdisent également les sous-tours. En effet, supposons que la solution entière possède un sous-tour $(p_1,\ldots,p_k,p_{k+1}=p_1)$ ne contenant pas la ville 0 alors

$$0 = w_{p_1} - w_{p_1} = \sum_{j=1}^{k}(w_{p_{j+1}} - w_{p_j}) \geqslant \sum_{j=1}^{k} t_{p_j p_{j+1}}$$

ce qui est absurde.

---

2 Modèles de graphes

Question 2 ***(8 points)***
Modéliser les problèmes suivants : nommer la classe de problème et définir l'instance.
Pour chaque problème, nommer un algorithme ou une méthode applicable à sa résolution.(*)

Problème 2.1. (1) *On dispose de 4 tâches à exécuter sur 4 machines. Chaque tâche i peut être effectuée par chaque machine j en un temps donné p_{ij}. Chaque machine ne reçoit qu'une tâche. On souhaite minimiser le temps total d'exécution des tâches.*

Problème 2.2. (1) *Un séminaire est organisé et p entreprises y participent ; chaque entreprise i envoie m_i représentants. Lors de ce séminaire, q groupes de travail sont menés simultanément ; chaque groupe j peut recevoir au plus n_j participants. Les organisateurs veulent répartir les participants dans les groupes de manière à ce que deux participants d'une même entreprise ne soient pas dans le même groupe. (Les groupes n'ont pas besoin d'être remplis.)*

Problème 2.3. (1) *La robustesse d'un réseau de télécommunication (non-orienté et connexe) se mesure en le nombre maximal de câbles pouvant simultanément tomber en panne sans déconnecter le réseau. On souhaite connaître, pour toute paire (u,v) de sommets du réseau, le nombre maximal $\kappa(u,v)$ de câbles pouvant être supprimés du réseau sans déconnecter u et v.*

Problème 2.4. (2) *Le plateau d'une usine de fabrication automatisée est équipée de rails sur lesquels des robots circulent. Les rails forment une grille carrée recouvrant le plateau : n rails orientés Est-Ouest et n rails orientés Nord-Sud. Une porte est située aux deux extrémités de chacun des rails. Les robots traversent le plateau en circulant sur les rails, sachant passer d'une direction à l'autre à toute intersection. On considère p robots placés sur p intersections de rails. En cas d'alerte, les p robots doivent sortir du plateau par n'importe quelle porte, en suivant des trajets qui ne se coupent pas (on ne se préoccupe pas des longueurs des trajets). Décidez si l'évacuation est possible.*

Problème 2.5. (3) *De nombreux phénomènes physiques sont représentables par des fonctions linéaires par morceaux. Une telle fonction* $f : [x_1, x_n] \subseteq \mathbb{R} \to \mathbb{R}$ *peut être définie par la séquence des points de brisure* $a_1 = (x_1, f(x_1)), \ldots, a_n = (x_n, f(x_n))$ *telle que* $x_1 < x_2 < \ldots < x_n$ *et* f *est linéaire sur chaque intervalle* $[x_i, x_{i+1}]$. *Si le nombre* n *de points est trop important, comme c'est le cas des phénomènes réels, la fonction ne peut être traitée efficacement. On souhaite alors approximer la fonction* f *par une seconde fonction linéaire par morceaux* g *définie par une sous-séquence de points* $a_1 = a_{k_1}, a_{k_2}, \ldots, a_{k_p} = a_n$ *de plus petite taille* ($p << n$). *L'erreur dûe à l'approximation est évaluée par* $c^g_- = \sum_{i=1}^{n} \big(f(x_i) - g(x_i)\big)^2$. *Le gain apporté par la diminution du nombre de points est évalué par* $c^g_+ = n - p$ *où* n *et* p *sont le nombre de points de* f *et de* g *respectivement. On cherche ainsi la meilleure approximation possible* g, *c'est à dire une approximation qui minimise* $c^g = \alpha c^g_- - \beta c^g_+$ *avec* α *et* β *des constantes positives quelconques.*
Note : $g(x_i)$ est calculable en tout point x_i. En effet, $g(x_i) = (x_i - x_{k_j})(f(x_{k_{j+1}}) - f(x_{k_j}))/(x_{k_{j+1}} - x_{k_j})$ pour tout $i \in [k_j, k_{j+1}]$.

1. Problème d'affectation (linéaire) = problème de couplage de coût minimal dans le graphe biparti pondéré $G = (T \cup M, T \times M, p)$. Algorithme hongrois $O(|T|^3)$.
2. On cherche un flot maximum de s vers t dans le graphe orienté $G = (V, E = E_1 \cup E_2 \cup E_3)$ avec capacité maximum $u \in \mathbb{R}_+^E$ sur les arcs (capacité minimum nulle) où : $V = \{s, t\} \cup \{e_1, \ldots, e_p\} \cup \{g_1, \ldots, g_p\}$, $(E_1, u) = \{((s, e_i), m_i) \ |i = 1..p\}$ $(E_2, u) = \{((e_i, g_j), 1) \ |i = 1..p, j = 1..q\}$ $(E_3, u) = \{(g_j, t), n_j) \ |j = 1..q\}$. Le problème est réalisable ssi la valeur de ce flot max est égale à $\sum_{i=1}^{p} m_i$. Algorithme de Ford-Fulkerson (chemin augmentant), Edmond-Karp (plus court chemin augmentant $O(|V||E|^2)$), Dinits (plus courts chemins augmentants $O(|V|^2|E|)$ ou $O(|V|^3)$), pré-flots, etc.
3. $\kappa(u, v) + 1$ est égal à la capacité minimum d'une (u, v)-coupe dans le graphe G formé par les arêtes du réseau pondérées par 1. Mêmes algorithmes que pour le flot max... les capacités unitaires réduisent la complexité (ex : $O(|V||E|)$).
4. On considère le graphe non-orienté G formé par la grille : les sommets sont les intersections de rails plus les portes, les arêtes sont les tronçons de rail entre deux sommets. On duplique chaque sommet-intersection de G et on ajoute un arc entre chaque sommet et son dupliqué. On ajoute un noeud fictif s et un arc de s vers chaque sommet-robot dans G. On ajoute un noeud fictif t et un arc entre chaque sommet-porte et t. Tous les arcs ont une capacité de 1. L'évacuation est possible ssi la valeur du (s, t)-flot max dans ce graphe est égale à p.
5. On considère le graphe orienté pondéré $G = (V, E, c)$ avec $V = \{a_1, \ldots, a_n\}$, $E = \{(a_i, a_j) | i < j\}$ et $c_{ij} = \alpha \sum_{k=i+1}^{j} (u_k^{ij})^2 - \beta(j - i - 1)$ où $u_k^{ij} = f(x_k) - ((x_k - x_i)(f(x_j) - f(x_i))/(x_j - x_i)$. On cherche un plus court chemin g de a_1 à a_n dans G : on note $k_j = i$ si a_i est le j-ème sommet de g, et p le nombre de sommets de g. Alors $u_k^{k_j k_{j+1}} = f(x_k) - g(x_k)$ pour tout $i \in [k_j, k_{j+1}]$ et le coût de g est égal à

$$\begin{aligned}
\sum_{j=1}^{p-1} c_{k_j k_{j+1}} = & \quad \sum_{j=1}^{p-1} \big(\alpha \sum_{k=k_j+1}^{k_{j+1}} (f(x_k) - g(x_k))^2 - \beta(k_{j+1} - k_j - 1)\big) \\
= & \quad \sum_{k=2}^{n} \alpha(f(x_k) - g(x_k))^2 - \beta(k_n - k_1 - (p-1)) \\
= & \quad \sum_{k=1}^{n} \alpha(f(x_k) - g(x_k))^2 - \beta(n - p)
\end{aligned}$$

Les coûts sont potentiellement négatifs mais le graphe est acyclique donc l'algorithme de Dijkstra ($O(|V|^2)$ s'applique : on augmente le coût de chaque arc par $c = \max_{(i,j)\in E}\{\max(0, -c_{ij})\}$.

3 Recherche locale : problème d'affectation quadratique

Question 3 *(8 points)*

Problème 3.1. n *unités communiquant entre elles doivent être placées sur* n *sites distants. On note* $f_{ij} \geqslant 0$ *le flux d'échanges entre deux unités* i *et* j *et* $d_{kl} \geqslant 0$ *la distance entre deux sites* k *et* l*. Le coût d'interaction du placement des unités* i *et* j *respectivement sur les sites* k *et* l *est égal au produit du flux entre les unités par la distance entre les sites :*

$$c_{ij} = f_{ij} d_{kl}.$$

Le problème d'affectation quadratique (QAP) consiste à trouver une affectation des unités aux sites (une unité par site et un site par unité) qui minimise la somme des coûts d'interaction sur toutes les paires d'unités.

Q3.1. (1) exprimer toute solution réalisable du QAP en terme de permutation ; dénombrer l'ensemble des solutions réalisables du QAP.

Q3.2. (1) exprimer la valeur du coût d'une solution réalisable du QAP par une formule mathématique, sachant que les matrices de flux f et de distance d sont symétriques.

Q3.3. (2) décrire une méthode constructive (algorithme glouton) **simple** pour le QAP et donner sa complexité.

Q3.4. (1) *voisinage sur les permutations :* décrire les mouvements de recherche locale consistant à échanger deux arêtes disjointes :

1. d'un tour dans le contexte du TSP (mouvement 2-opt)
2. d'une affectation dans le contexte du QAP (mouvement transposition)

Donner la taille du voisinage de ce dernier.

Q3.5. * On note p une solution réalisable, $p(k)$ le site affecté à une unité k ($1 \leqslant k \leqslant n$), et p_{ij} la solution obtenue après échange des placements de deux unités i et j ($1 \leqslant i < j \leqslant n$). Montrer que la variation du coût occasionnée par cette échange est égale à :

$$\Delta_{ij}^{p} = 2 \sum_{k=1, k\neq i, k\neq j}^{n} (f_{ik} - f_{kj})(d_{p(k)p(j)} - d_{p(k)p(i)}) \tag{9}$$

Q3.6. * Soient deux paires distinctes d'unités (u, v) et (i, j), une solution réalisable p, et p_{uv} la solution obtenue depuis p par échange des placements de u et v. Montrer que :

$$\Delta_{ij}^{p_{uv}} - \Delta_{ij}^{p} = 2\big((f_{iu} - f_{uj}) - (f_{iv} - f_{vj})\big)\big((d_{p(v)p(j)} - d_{p(v)p(i)}) + (d_{p(u)p(i)} - d_{p(u)p(j)})\big) \tag{10}$$

Q3.7. (3) À partir des formules des deux questions précédentes, écrire l'algorithme d'une méthode de descente pour le QAP ; et donner sa complexité.

1. On numérote les unités de 1 à n et les sites de 1 à n, alors à toute solution réalisable du QAP correspond une unique permutation p de l'ensemble $\{1, \ldots, n\}$ telle que $p(i)$ désigne le numéro du site affecté à la i-ème unité. L'ensemble des solutions réalisables du QAP est donc de cardinalité $n!$ (nombre de permutations possibles).
2. on minimise $2c(p)$ sur l'ensemble des permutations p avec $c(p) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} f_{ij} d_{p(i)p(j)}$.
3. l'heuristique la plus simple ($O(n)$) consiste à poser $p(i) = i$ pour tout $i = 1..n$. Une autre heuristique gloutonne consiste à affecter les paires d'unités avec les coûts d'interactions les plus faibles en premier :

step 0 on trie la liste F des $(f_{ij}|1 \leqslant i < j \leqslant n)$ par ordre croissant
on trie la liste D des $(d_{kl}|k,l = 1..n)$ par ordre croissant

step 1 on cherche le prochain f_{ij} dans F tel que ni i ni j ne sont affectés
si f_{ij} trouvé, on cherche le prochain d_{kl} dans D tel que ni k ni l ne soient affectés
si d_{kl} trouvé, on pose $p(i) = k$ et $p(j) = l$ puis on réitère step 1.

step 2 à chaque unité restante non affectée, on associe un site restant non affecté.

La complexité est $O(n^2)$ (step 0 : $n^2 + n^2$ step 1 itéré : $n^2 + n^2$, step 2 : $n + n$).

4. Un tour et une affectation peuvent tous deux être encodés par une permutation $p = (p(1), \ldots, p(n))$: le tour est la séquence des arêtes $(p(i), p(i+1))$ du graphe (avec $p(n+1) = p(1)$) et l'affectation est l'ensemble des arêtes du couplage biparti $(i, p(i))$ (avec i l'unité et $p(i)$ le site).
   (a) dans le contexte du TSP : l'échange de deux arêtes disjointes $(p(i), p(i+1))$ et $(p(j), p(j+1))$ mène à un nouveau tour correspondant à la permutation
   $$(p(1), \ldots, p(i), p(j), p(j-1), \ldots, p(i+1), p(j+1), \ldots, p(n))$$
   (b) dans le contexte du QAP : l'échange de deux arêtes disjointes $(i, p(i))$ et $(j, p(j))$ mène à une nouvelle affectation correspondant à la permutation
   $$(p(1), \ldots, p(i-1), p(j), p(i+1), \ldots, p(j-1)p(i), p(j+1), \ldots, p(n))$$

   Ainsi dans le contexte du QAP, pour chaque permutation p et pour chaque paire d'unités $1 \leqslant i < j \leqslant n$, on associe la permutation :
   $$p_{ij}(u) = \begin{cases} p(j) & \text{si } u = i \\ p(i) & \text{si } u = j \\ p(u) & \text{sinon.} \end{cases}$$
   Le mouvement de transposition est l'ensemble des opérations $o_{ij} : p \mapsto p_{ij}$ avec $1 \leqslant i < j \leqslant n$. Comme $p(u) \neq p(v)$ pour tout $u \neq v$, on montre facilement que $p_{ij} = p_{i'j'} \Rightarrow (i,j) = (i',j')$. Autrement dit, les opérations o_{ij} définissent des permutations toutes différentes. Donc
   $$card(V(p)) = card\{(i,j)|1 \leqslant i < j \leqslant n\} = \sum_{i=1}^{n-1}(n-i) = \sum_{i=1}^{n-1} i = n(n-1)/2$$

5. $\Delta^p_{ij} = 2c(p_{ij}) - 2c(p) = 2\sum_{u=1}^{n-1}\sum_{v=u+1}^{n} f_{uv}(d_{p_{ij}(u)p_{ij}(v)} - d_{p(u)p(v)})$ On note a^{ij}_{uv} chaque terme de cette somme. Seuls les termes tels que $u \in \{i,j\} \iff v \notin \{i,j\}$ sont non nuls, c'est à dire tous les termes a^{ij}_{uv} tels que :
   $$(u,v) \in \{(k,i),(k,j)|k = 1..i-1\} \cup \{(i,k),(k,j)|k = i+1..j-1\} \cup \{(i,k),(j,k)|k = j+1..n\}$$
   Ainsi : $\Delta^p_{ij} = 2\sum_{k=1,k\neq i,k\neq j}^{n}(a^{ij}_{ik} + a^{ij}_{kj})$ avec $a^{ij}_{ik} + a^{ij}_{kj} = (f_{ik} - f_{kj})(d_{p(k)p(j)} - d_{p(k)p(i)})$

6.
$$\begin{aligned}(\Delta^{p_{uv}}_{ij} - \Delta^p_{ij})/2 = & \sum_{k=1,k\neq i,k\neq j}^{n}(f_{ik} - f_{kj})\big((d_{p_{uv}(k)p_{uv}(j)} - d_{p_{uv}(k)p_{uv}(i)}) - (d_{p(k)p(j)} - d_{p(k)p(i)})\big) \\ = & (f_{iu} - f_{uj})\big((d_{p(v)p(j)} - d_{p(v)p(i)}) - (d_{p(u)p(j)} - d_{p(u)p(i)})\big) + \\ & (f_{iv} - f_{vj})\big((d_{p(u)p(j)} - d_{p(u)p(i)}) - (d_{p(v)p(j)} - d_{p(v)p(i)})\big) \\ = & \big((f_{iu} - f_{uj}) - (f_{iv} - f_{vj})\big)\big((d_{p(v)p(j)} - d_{p(v)p(i)}) + (d_{p(u)p(i)} - d_{p(u)p(j)})\big)\end{aligned}$$

7. on définit les fonctions suivantes :
   `cout()` { RETURN $\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} f_{ij}d_{ij}$ }
   `delta(i,j)` { RETURN $\sum_{k=1,k\neq i,k\neq j}^{n}(f_{ik}-f_{kj})(d_{kj}-d_{ki})$ }
   `delta2(i,j,u,v)` { RETURN $\big((f_{iu}-f_{uj})-(f_{iv}-f_{vj})\big)\big((d_{vj}-d_{vi})+(d_{ui}-d_{uj})\big)$ }
   `swap(p,u,v)` { $k = p[u]$; $p[u] = p[v]$; $p[v] = k$ }
   Elles sont de complexités respectives $O(n^2)$, $O(n)$, $O(1)$, et $O(1)$.

   step 0. initialisation : $p[i] = i$ pour $i = 1..n$
   `c =cout()`
   $D[i][j] =$ `c+delta(i,j)` pour $i = 1..n-1, j = i+1..n$

   step 1. déterminer (u, v, c') tels que $c' = max_{(i,j)} D[i][j] = D[u][v]$

   step 2. si $c' \geqslant c$ **STOP RETURN** p

   step 3. `swap(p,u,v)`
   $D[i][j] =$ `D[i][j]+delta2(i,j,u,v)` pour $i = 1..n-1, j = i+1..n$
   GOTO step 1

   L'initialisation est réalisée en $O(n^3)$ et chaque itération en $O(n^2)$.

---

4 Dualité : flot max / coupe min

L'objectif de cet exercice est de (re)démontrer la relation de dualité entre le problème de flot maximum et le problème de coupe de capacité minimum.

Dans cette étude, on considère $G = (V, E, c)$ un graphe orienté muni d'une capacité $c_e \in \mathbb{Z}_+$ sur chaque arc $e \in E$. On suppose que G possède deux sommets $s \in V$ et $t \in V$ liés par un arc $(t, s) \in E$ et tels que (t, s) soit l'unique arc entrant en s et l'unique arc sortant de t.

Un *flot* $f \in \mathbb{Z}_+^E$ désignera ci-après un flot de s à t compatible dans G. Une *coupe* $W \subsetneq V$ désignera une (s,t)-coupe dans G ($s \in W$ et $t \in \bar{W} = V \setminus W$) de capacité $\sum_{e\in E\cap W\times\bar{W}} c_e$.

Question 4 ***(8 points)***

Q4.1. (0,5) donner la classe de complexité du problème de flot maximum.

Q4.2. (0,5) donner une condition nécessaire et suffisante de réalisablité du problème de flot maximum dans G.

Q4.3. (1) formuler le problème de flot maximum dans G par un programme linéaire de la forme :

$$(P) : \max f_{ts} \quad \text{s.t. } Af = 0, f \leqslant b, f \in \mathbb{R}_+^E$$

Définir les éléments A et b dans cette formulation.

Q4.4. (1) écrire le programme linéaire dual (D) de (P).

Q4.5. (1) soit une (s,t)-coupe $W \subseteq V$ dans G ; montrer que la solution $(x, y) \in \mathbb{R}^V \times \mathbb{R}^E$ définie ci-dessous est une solution réalisable de (D) de coût égal à la capacité de la coupe W :

$$x_i = \begin{cases} 1 & \text{si } i \in \bar{W} \\ 0 & \text{si } i \in W. \end{cases} \tag{11}$$

$$y_{ij} = \begin{cases} 1 & \text{si } i \in W \text{ et } j \in \bar{W} \\ 0 & \text{sinon.} \end{cases} \tag{12}$$

En déduire que le problème de flot maximum dans G et le problème de coupe de capacité minimum dans G vérifient la propriété de dualité faible.

Q4.6. (2) soient f un (s,t)-flot compatible dans G et W une (s,t)-coupe dans G ; montrer que f et W sont conjointement optimums (flot maximum et capacité minimum) si :

$$f_{ij} = 0 \qquad \forall (i,j) \in (\bar{W} \times W) \setminus \{(t,s)\} \tag{13}$$

$$f_{ij} = c_{ij} \qquad \forall (i,j) \in W \times \bar{W} \tag{14}$$

Q4.7. (2) montrer que l'implémentation de l'algorithme de Ford-Fulkerson basée sur la recherche d'une chaîne augmentante (voir Algorithme 2) construit à la dernière itération (step 4) une (s,t)-coupe de capacité égale à la valeur du flot max. En déduire que le problème de flot maximum dans G et le problème de coupe de capacité minimum dans G vérifient la propriété de dualité forte.

Remarque : Il n'y pas de stricte équivalence entre le programme linéaire (D) *et le problème de coupe min. On fera donc attention à ne pas confondre les propriétés vérifiées par* (D) *avec les propriétés vérifiées par le problème de coupe min.*

1. L'algorithme de Edmond-Karp, par exemple, permet de résoudre le problème de flot max en temps polynomial $O(VE^2)$. Le [problème de décision associé au] problème d'optimisation du flot max appartient donc à la classe $\mathcal{P}$.
2. Le flot identiquement nul est un flot compatible de G car $c \geqslant 0$. Le problème de flot max est donc toujours réalisable dans G.
3. $A \in \mathbb{Z}_+^V \times E$ est la matrice d'incidence du graphe : $a_{ie} = 1$ si $e \in \{i\} \times V$, $a_{ie} = -1$ si $e \in V \times \{i\}$, $a_{ie} = 0$ sinon. $b = c \in \mathbb{Z}_+^E$ est le vecteur des capacités :

$$\begin{array}{lll} \max & f_{ts} & \\ \text{s.t.} & \sum_{e \in E} a_{ie} f_e = 0 & i \in V \\ & f_e \leqslant c_e & \forall e \in E \\ & f_e \in \mathbb{R}_+ & \forall e \in E \end{array}$$

4. Dual :

$$\begin{array}{lll} \min & \sum_{e \in E} c_e y_e & \\ \text{s.t.} & x_t - x_s + y_{ts} \geqslant 1 & \\ & x_i - x_j + y_{ij} \geqslant 0 & \forall (i,j) \in E \\ & x_i \in \mathbb{R} & \forall i \in V \\ & y_e \in \mathbb{R}_+ & \forall e \in E \end{array}$$

5. Réalisabilité : pour tout arc $(i,j) \in E$,

$$x_i^W - x_j^W + y_{ij}^W = \begin{cases} 1-0+0=1 & \text{si } (i,j) \in \bar{W} \times W \text{ (arc arrière)} \\ 0-1+1=0 & \text{si } (i,j) \in W \times \bar{W} \text{ (arc avant)} \\ 0-0+0=0 & \text{si } (i,j) \in W \times W \\ 1-1+0=0 & \text{si } (i,j) \in \bar{W} \times \bar{W} \end{cases}$$

   Coût : $\sum_{e \in E} c_e y_e^W = \sum_{e \in E \cap W \times \bar{W}} c_e$ = capacité de la coupe W.
6. (P) et (D) vérifient la propriété de dualité faible (et forte) donc toute solution réalisable de (D) – et a fortiori, toute solution de (D) associée à une coupe W – a un coût supérieur ou égal à toute solution de (P), donc à tout (s,t)-flot compatible de G.
7. Soit f un flot et W une coupe. Le théorème des écarts complémentaires appliqué à (P) et (D) se traduit par : f et (x^W, y^W) sont des solutions optimales de (P) et (D) respectivement si et

seulement si :

$$f_{ts}(x_s^W - x_t^W + y_{ts}^W - 1) = 0 \tag{15}$$

$$f_{ij}(x_i^W - x_j^W + y_{ij}^W) = 0 \qquad \forall (i,j) \in E' = E \setminus \{(t,s)\} \tag{16}$$

$$(\sum_{e \in E} a_{ie} f_e) x_i = 0 \qquad \forall i \in V \tag{17}$$

$$(c_e - f_e) y_e^W = 0 \qquad \forall e \in E \tag{18}$$

$$\tag{19}$$

La première condition est vérifiée pour toute coupe W (car $x_s^W - x_t^W + y_{ts}^W = 1$). La seconde condition est vérifiée pour tout arcs $(i,j) \notin \bar{W} \times W$ (car $x_i^W - x_j^W + y_{ij}^W = 0$). Elle est vérifiée pour $(i,j) \in (\bar{W} \times W) \setminus \{(t,s)\}$ si et seulement si $f_{ij} = 0$ (car $x_i^W - x_j^W + y_{ij}^W = 1$). La troisième condition est vérifiée pour tout flot réalisable f (car $\sum_{e \in E} a_{ie} f_e = 0$). La quatrième condition est vérifiée pour tout $e \notin W \times \bar{W}$ (car $y_e^W = 0$). Elle est vérifiée pour $e \notin W \times \bar{W}$ si et seulement si $f_e = c_e$ (car $y_e^W = 1$).
Par ailleurs, W est une coupe de capacité minimale si (x^W, y^W) est une solution optimale de (D) (remarque : nous n'avons pas encore prouvé l'inverse car nous n'avons pas encore prouvé qu'il existe une coupe dont la solution associée soit optimale dans (D)).

8. Soit f le flot (maximal) retourné par l'algorithme et soit W l'ensemble des sommets marqués à l'issue de la dernière itération de l'algorithme. $s \in W$ et $t \notin W$ donc W est une (s,t)-coupe. Par construction, pour tout arc $e = (i,j) \in E$, on a :

$$\begin{cases} 0 < f_{ij} < c_{ij} & \text{si } (i,j) \in W \times W \\ f_{ij} = 0 & \text{si } (i,j) \in (\bar{W} \times W) \setminus \{(t,s)\} \text{ (arc arrière)} \\ f_{ij} = c_{ij} & \text{si } (i,j) \in W \times \bar{W} \text{ (arc avant)} \end{cases}$$

Ainsi, f et (x^W, y^W) vérifient les conditions de la question précédente et sont donc optimales pour (P) et (D). Comme (P) et (D) vérifient la propriété de dualité forte, leurs coûts sont égaux et donc W est une coupe de capacité égale à la valeur du flot max f. Comme la capacité de toute coupe est supérieure ou égale à la valeur de tout flot, la coupe W est nécessairement de capacité minimale.

Annexe : aide-mémoire

Définition 1 *Dualité faible/forte.*
Deux problèmes d'optimisation combinatoire P (problème de minimisation) et D (problème de maximisation) vérifient la propriété de **dualité faible** si le coût de toute solution réalisable dans D est inférieure ou égale au coût de toute solution réalisable de P. Si de plus, P et D possèdent des solutions réalisables dont les coûts respectifs sont identiques, alors P et D vérifient la propriété de **dualité forte**.

Théorème 1 *Écarts complémentaires.*
Soient deux programmes linéaires $(P) : \min\{ cx \mid Ax \geqslant b, x \in \mathbb{R}^n_+ \}$ et $(D) : \max\{ by \mid yA \leqslant c, y \in \mathbb{R}^m_+ \}$ définis par $c \in \mathbb{Z}^n_+$, $b \in \mathbb{Z}^m_+$ et $A \in \mathbb{Z}^{m\times n}_+$ et soient, respectivement, deux solutions réalisables $\bar{x}$ et $\bar{y}$. Les solutions $\bar{x}$ et $\bar{y}$ sont optimales respectivement dans (P) et dans (D) si et seulement si :

$$y_i(\sum_{j=1}^{n} a_{ij}x_j - b_i) = 0 \qquad \forall i = 1,\dots,m \tag{20}$$

$$(c_j - \sum_{i=1}^{m} y_i a_{ij})x_j = 0 \qquad \forall j = 1,\dots,n \tag{21}$$

De plus (P) et (D) vérifient la propriété de dualité forte.

Algorithme 1 *Ford Fulkerson (chaîne augmentante).*
Soit un graphe orienté $G = (V, E, c)$ muni d'une capacité $c_e \in \mathbb{Z}_+$ sur chaque arc $e \in E$. G possède deux sommets $s \in V$ et $t \in V$ liés par un arc $(t, s) \in E$ et tels que (t, s) soit l'unique arc entrant en s et l'unique arc sortant de t. On note $E' = E \setminus \{(t, s)\}$.

step 0. initialisation : f est le flot nul sur G
step 1. démarquer tous les sommets sauf s
step 2. si tous les sommets marqués ont été traités : **GOTO step 4**
step 3. choisir un sommet marqué non-traité i
marquer j pour tout arc sortant $(i, j) \in E'$ tel que $f_{ij} < c_{ij}$
marquer j pour tout arc entrant $(j, i) \in E'$ tel que $f_{ji} > 0$
step 4. si t n'est pas marqué **STOP RETURN** f maximal
step 5. **augmenter**[a] f le long d'une chaîne marquée de s à t et **GOTO step 1**

[a] algorithme [de l'augmentation du flot] non détaillé ici.

Définition 2 *Mouvement de recherche locale/voisinage.*
Soit E l'espace de recherche d'un problème d'optimisation combinatoire (P) et soit $S \subseteq E$ l'ensemble des solutions réalisables de (P). Un **mouvement de recherche locale** est un ensemble d'opérations $\{f_x : E \to E \mid x \in X\}$ qui à toute solution $s \in S$ associe une solution réalisable $f_x(s) \in S$. Le **voisinage** d'une solution réalisable $s \in S$ est l'image de s par cet ensemble d'opérations $V(s) = \{f_x(s) \mid x \in X\} \subseteq S$.

Algorithme 2 *Méthode de descente.*
Soit P un problème d'optimisation combinatoire de la forme $\min\{f(s) \mid s \in S\}$. On note $V(s)$ le voisinage d'une solution $s \in S$ pour un mouvement de recherche locale donné.

step 0. initialisation : $s \in S$ solution réalisable
step 1. déterminer un optimum local : une solution $s' \in V(s)$ qui minimise f
step 2. si $f(s') \geqslant f(s)$ **STOP RETURN** s
step 3. poser $s = s'$ et **GOTO step 1**