Algorithmique
Magistère STIC 2002/2003

Notes de cours

S. Demri (LSV, ENS de Cachan)

Avril 2003

Cette partie du cours d'algorithmique traite
- du problème du flot maximal [CLR94, Chapitre 27] ;
- NP-complétude [CLR94, Chapitre 36] ;
- des algorithmes d'approximation [CLR94, Chapitre 37].

Ces notes sont fortement inspirées (mais pas toujours) de certaines sections de :

Th. Cormen, Ch. Leiserson, and R. Rivest. *Introduction à l'algorithmique*. Dunod, 1994.

L'envoi de commentaires, typos, ... est bienvenu (demri@lsv.ens-cachan.fr).

Table des matières

1 Le problème du flot maximal

Problèmes qui mettent en jeu un flux physique de matière, électricité, ou d'information.

Problème de flux maximal du TD de Calculabilité de Marie Duflot.

1.1 Réseaux de transport

Définition 1.1. (réseau de transport) $G = \langle S, A, s, t, c \rangle$
- $G = \langle S, A \rangle$ graphe orienté
- $s \in S$ (source) ;
- $t \in S$ (puits) ;
- $c : S \times S \to \mathbb{N} \cup \{\infty\}$ (capacité),
  $c(u, v) = 0$ ssi $(u, v) \notin A$.

Exemple 1.1. (réseau de transport) Exemple 27.1(a) de [CLR94, page 570].

Définition 1.2. (flot) $f : S \times S \to \mathbb{R}$ vérifiant :

(capacité) : $\forall u, v \in S$, $f(u, v) \leq c(u, v)$;

(symétrie) : $\forall u, v \in S$, $f(u, v) = -f(v, u)$;

(conservation) : $\forall u \in S \setminus \{s, t\}$, $\sum_{v \in S} f(u, v) = 0$.

Définition 1.3. (valeur d'un flot) La valeur du flot f, notée $|f|$, est $|f| = \sum_{v \in S} f(s, v)$.

Définition 1.4. (problème du flot maximal)

entrée : un réseau de transport (sans chemin de s à t étiqueté seulement par ∞) ;

sortie : un flot de valeur maximale entre s et t.

Borne supérieure de la valeur du flot maximal sur un réseau à capacités finies avec $f : S \times S \to \mathbb{Z}$ (au lieu de $\mathbb{R}$) : $\sum_{(u,v) \in A} c(u, v)$.

Exemple 1.2. (flot) Exemple 27.1(b) de [CLR94, page 570].

Notation. (sommation implicite) Si X et Y sont des ensembles de sommets (de S), alors

$$f(X,Y) = \sum_{x\in X}\sum_{y\in Y} f(x,y).$$

Lemme 1.3. Soient $G = \langle S, A, s, t, c\rangle$ un réseau de transport et f un flot de G.

(I) pour $X \subseteq S$, $f(X,X) = 0$;

(II) pour $X, Y \subseteq S$, $f(X,Y) = -f(Y,X)$;

(III) pour $X, Y, Z \subseteq S$ avec $X \cap Y = \emptyset$,

(III.1) $f(X \cup Y, Z) = f(X,Z) + f(Y,Z)$;

(III.2) $f(Z, X \cup Y) = f(Z,X) + f(Z,Y)$.

$$\begin{aligned}
|f| &= f(s,S)\\
&= f(S,S) - f(S\setminus\{s\},S) \text{ (lemme 1.3(III.1))}\\
&= f(S, S\setminus\{s\}) \text{ (lemme 1.3(I,II))}\\
&= f(S,t) + f(S, S\setminus\{s,t\}) \text{ (lemme 1.3(III.2))}\\
&= f(S,t) \text{ (par conservation du flot).}
\end{aligned}$$

Exercice 1.1. Exercices 27.1.6 (somme des flots) et 27.1.7 (produit scalaire-flot) de [CLR94, page 576]. “Les flots d’un réseau forment un ensemble convexe.”

Flot net positif rentrant de v :

$$\Sigma_{u \in S, f(u,v)>0} f(u, v).$$

Similaire pour le flot net sortant.

Convention : au plus un arc entre deux sommets pour le flot (par annulation). Figure 27.2 [CLR94, 573].

Exercice 1.2. Coder le problème suivant comme un problème de flot maximal.

1. $C_1, \ldots, C_n$: centres de distribution avec quantités disponibles $c_1, \ldots, c_n$;
2. $D_1, \ldots, D_m$: demandes faites par des clients de quantités $d_1, \ldots, d_m$;
3. chaque client ne peut être livré que par un sous-ensemble de centres ;
   $R \subseteq \{C_1, \ldots, C_n\} \times \{D_1, \ldots, D_m\}$;
4. Optimiser l'approvisionnement de l'ensemble des clients.

1.2 Modèle étendu des réseaux

Définition 1.5. (réseau de transport à sources et puits multiples) graphe orienté $G = \langle S, A \rangle$ muni de
- $s_1, \ldots, s_m \in S$ (sources) ;
- $t_1, \ldots, t_n \in S$ (puits) ;
- $c : S \times S \to \mathbb{N} \cup \{\infty\}$.

Exercice 1.3. Comment réduire le problème du flot maximal sur un réseau de transport à sources et puits multiples en un problème de flot maximal sur un réseau de transport standard (cf. [CLR94, page 574, fig. (b)]).

Exercice 1.4. Ajout de capacité aux noeuds de réseau [Gar00]. Pour $u \in S$, on ajoute une capacité $c(u) \in \mathbb{N}$ et le flot doit vérifier la condition :

(capacité noeud) : $\sum_{(u,v)\in A, f(u,v)>0} f(u,v) \leq c(u), \forall u \in S$.

Montrer comment réduire le problème du flot maximal avec capacité sur les noeuds en un problème de flot maximal sur un réseau de transport standard (solution dans [Gar00]).

Définition 1.6. (problème du flot maximal avec coût minimal) [Gar00]. Ajout du coût $ct(u, v)$ pour $(u, v) \in A$. Recherche du flot maximal avec coût minimal.

1.3 La méthode de Ford-Fulkerson

Définition 1.7. (taille d'un réseau)

$$|G| = |A|(1 + log(max(c(u, v))) + |S|.$$

Les entiers sont codés en binaire.

Exercice 1.5. Donner un algorithme pour résoudre le problème du flot maximal quand toutes les capacités sont finies et les flots sont entiers. Quelle la complexité en temps de calcul en fonction de $|G|$?

But de la section : calcul du flot maximal en temps polynomial en $|G|$ (attention on prend le logarithme des $c(u, v)$ dans le calcul de $|G|$).

Idée de la méthode de Ford-Fulkerson :
– réseaux résiduels ;
– amélioration des chemins ;
– coupe minimale.

MÉTHODE-FORD-FULKERSON(G, s, t)

1. initialiser le flot f à 0 ;
2. TANT QUE il existe un chemin améliorant p
3. FAIRE augmenter le flot f le long de p
4. RETOURNER f

Définition 1.8. (capacité résiduelle, réseau résiduel) Soient G un réseau de transport et f un flot. La capacité résiduelle de (u, v) est :

$$c_f(u, v) = c(u, v) - f(u, v).$$

Le réseau résiduel de G induit par f est $G_f = \langle S, A_f, s, t, c_f \rangle$ avec

$$A_f = \{(u, v) \in S \times S : c_f(u, v) > 0\}.$$

Exemple 1.4. Graphe résiduel de [CLR94, page 570, fig. (b)].

Lemme 1.5. Soient $G = \langle S, A, s, t, c\rangle$ un réseau de transport et f un flot de G. Soit G_f le réseau résiduel de G induit par f, et soit f' un flot de G_f. Alors la somme $f + f'$ est un flot de G de valeur $|f + f'| = |f| + |f'|$.

Par définition $(f + f')(u, v) = f(u, v) + f'(u, v)$.

Preuve. Vérification des propriétés de symétrie, capacité, et conservation.
(symétrie)

$$\begin{aligned}(f + f')(u, v) &= f(u, v) + f'(u, v)\\ &= -f(v, u) + -f'(v, u)\\ &= -(f(v, u) + f'(v, u))\\ &= -(f + f')(v, u).\end{aligned}$$

(capacité) $f'(u, v) \leq c_f(u, v) = c(u, v) - f(u, v)$.

$$\begin{aligned}(f + f')(u, v) &= f(u, v) + f'(u, v)\\ &\leq f(u, v) + (c(u, v) - f(u, v))\\ &\leq c(u, v).\end{aligned}$$

(conservation) $u \in S \setminus \{s, t\}$.

$$\begin{aligned}\textstyle\sum_{v\in S}(f + f')(u, v) &= \textstyle\sum_{v\in S}(f(u, v) + f'(u, v))\\ &= \textstyle\sum_{v\in S} f(u, v) + \sum_{v\in S} f'(u, v)\\ &= 0 + 0 = 0.\end{aligned}$$

De plus, on a

$$
\begin{aligned}
|f + f'| &= \sum_{v \in S} (f + f')(s, v) \\
&= \sum_{v \in S} f(s, v) + f'(s, v) \\
&= \sum_{v \in S} f(s, v) + \sum_{v \in S} f'(s, v) \\
&= |f| + |f'|.
\end{aligned}
$$

C.Q.F.D.

Définition 1.9. (chemin améliorant, capacité résiduelle) Soient G un réseau de transport et f un flot. Un chemin améliorant p est un chemin simple (cad sans répétition de noeuds) de s vers t dans le réseau résiduel G_f. La capacité résiduelle de p est :

$$c_f(p) = min\{c_f(u, v) : (u, v) \in p\}.$$

Lemme 1.6. Soient G un réseau de transport, f un flot, et p un chemin améliorant de G_f. La fonction f_p définie ci-dessous est un flot de G_f de valeur $|f_p| = c_f(p) > 0$:

$$
f_p(u, v) = \begin{cases}
c_f(p) \text{ si } (u, v) \text{ appartient à } p \\
-c_f(p) \text{ si } (v, u) \text{ appartient à } p \\
0 \text{ sinon.}
\end{cases}
$$

Preuve. Laissée en exercice. C.Q.F.D.

Corollaire 1.7. Soient G un réseau de transport, f un flot, et p un chemin améliorant de G_f. $f' = f + f_p$ est un flot de G de valeur supérieure à $|f|$.

Comment est-on assurer que s'il n'y a pas de chemin améliorant alors un flot maximal est atteint ?

Une coupe est une partition d'un ensemble de noeuds (cf. aussi la notion de coupe dans un arbre couvrant minimal).

Définition 1.10. (coupe, flot net, capacité, coupe minimum) Une coupe (E, T) d'un réseau de transport G est une partition de S telle que $s \in E$ et $t \in T$.
Flot net : $f(E, T)$.
Capacité : $c(E, T)$.
Coupe minimum : coupe dont la capacité est minimale rapportée à toutes les coupes du réseau.

Lemme 1.8. Soient G un réseau de transport, f un flot, et (E, T) une coupe de G. On a $f(E, T) = |f|$.

Preuve. Utilisation répétée du lemme 1.3.

$$\begin{aligned} f(E,T) &= f(E,S) - f(E,E) \\ &= f(E,S) \\ &= f(s,S) + f(E \setminus \{s\}, S) \\ &= f(s,S) + 0 \text{ (car } E \setminus \{s\} \subseteq S \setminus \{s,t\}) \\ &= |f|. \end{aligned}$$

C.Q.F.D.

Corollaire 1.9. La valeur d'un flot quelconque dans un réseau de transport G est bornée supérieurement par la capacité d'une coupe quelconque de G.

Preuve. (E, T) coupe quelconque de G et f flot quelconque de G.

$$\begin{aligned} |f| &= f(E,T) \\ &= \sum_{u\in E}\sum_{v\in T} f(u,v) \\ &\leq \sum_{u\in E}\sum_{v\in T} c(u,v) \\ &\leq c(E,T). \end{aligned}$$

C.Q.F.D.

Résultat majeur du problème du flot maximal :

Théorème 1.10. (flot maximal et coupe minimale) Si f est un flot dans un réseau de transport G alors les conditions suivantes sont équivalentes :

(I) f est un flot maximal dans G ;

(II) G_f ne contient aucun chemin améliorant ;

(III) $|f| = c(E, T)$ pour une certaine coupe (E, T) de G.

Preuve. (I) implique (II). Supposons que f est un flot maximal et que G_f contient un chemin améliorant p. D'après le corollaire 1.7, $f + f_p$ est un flot de G de valeur supérieure à celle de f, une contradiction.
(II) implique (III). Supposons que G_f ne contienne pas

de chemin améliorant, c'est-à-dire qu'il n'y a pas de chemin de s vers t dans G_f. Soit E le sous-ensemble de S composé des noeuds qui peuvent être atteints de s dans G_f et $T = S \setminus E$. (E, T) forme donc une coupe de G_f. Pour $u \in E$ et $v \in T$, on a $(u, v) \notin A_f$ (sinon on aurait $v \in E$). Par conséquent, $f(u, v) = c(u, v)$. D'après le lemme 1.8, $|f| = f(E, T) = c(E, T)$.
(III) implique (I). D'après le corollaire 1.9, $|f| \leq c(E, T)$ pour toutes les coupes (E, T). Par conséquent, $|f| = c(E, T)$ implique que f est un flot maximal.
C.Q.F.D.

FORD-FULKERSON(G, s, t)

1. POUR chaque arc $(u, v) \in A$
   FAIRE $f[u, v] \leftarrow 0$; $f[v, u] \leftarrow 0$;
2. TANT QUE il existe un chemin simple p de s vers t dans G_f
   FAIRE
   – $c_f(p) \leftarrow min\{c_f(u, v) : (u, v) \in p\}$
   – POUR chaque arc (u, v) de p
     FAIRE $f[u, v] \leftarrow f[u, v]+c_f(p)$; $f[v, u] \leftarrow -f[u, v]$.

Nombre de passages dans la boucle TANT QUE au plus $|f^*|$ où f^* est un flot maximal. Or $|f^*|$ est en $2^{\mathcal{O}(|G|)}$.

Exemple 1.11. Mauvais réseau, Figure 27.7 de [CLR94, page 586].

Définition 1.11. (algorithme de Edmonds-Karp) Méthode de Ford-Fulkerson avec la recherche d'un plus court chemin (chaque arc a une distance unitaire).

Algorithme de Dijkstra en temps d'exécution en $\mathcal{O}(|S|^2)$.

L'algorithme du plus court chemin avec parcours en largeur nécessite un temps de calcul en $\mathcal{O}(|S| + |A|)$ quand le graphe est représenté par des listes d'adjacence[CLR94, pages 462–463].

$\delta_f(u, v)$: distance entre u et v dans G_f où chaque arc possède une distance unitaire. $\delta_f(u, v) = \infty$ s'il n'y a pas de chemin.

Lemme 1.12. Si l'algorithme de Edmonds-Karp est exécuté sur un réseau G alors pour tous les sommets $v \in S \setminus \{s, t\}$, $\delta_f(s, v)$ augmente de façon monotone à chaque augmentation de flot.

Attention : pas d'augmentation stricte ! ! Si $\delta_f(u, v) \neq \infty$, alors $\delta_f(u, v) \leq |S| - 1$

Preuve. Supposons qu'il existe $v \in S \setminus \{s, t\}$ pour qui une augmentation du flot provoque une diminution de $\delta_f(s, v)$. Soit f le flot avant augmentation et f' le flot après. On a donc

$$\delta_{f'}(s, v) < \delta_f(s, v).$$

Sans perte de généralité, on suppose que pour $u \in S \setminus \{s, t\}$, $\delta_{f'}(s, u) < \delta_{f'}(s, v)$ implique $\delta_f(s, u) \leq \delta_{f'}(s, u)$.

Soit p' un plus court chemin dans $G_{f'}$ de la forme $s \xrightarrow{*} u \rightarrow v$. S'il n'y a pas de plus court chemin, $\delta_{f'}(s, v) = \infty$ et donc une contradiction est immédiate.

On peut montrer que $\delta_{f'}(s, u) = \delta_{f'}(s, v) - 1$ car p' est un plus court chemin et $\delta_f(s, u) \leq \delta_{f'}(s, u)$ par hypothèse.

Supposons que $f[u, v] < c(u, v)$ où $f[u, v]$ est le flot net de u vers v avant l'augmentation de flot dans G_f.

$$\begin{aligned}\delta_f(s,v) &\leq \delta_f(s,u)+1\\ &\leq \delta_{f'}(s,u)+1\\ &\leq \delta_{f'}(s,v),\end{aligned}$$

ce qui mène encore à une contradiction.

Par conséquent $f[u,v] = c(u,v)$ et donc $(u,v) \notin A_f$. Le chemin p qui est choisi dans G_f pour produire $G_{f'}$ doit contenir (v,u) puisque $(u,v) \in A_{f'}$ et $(u,v) \notin A_f$. Le choix de p provoque un reflux le long de (u,v) : v apparaît avant u sur le chemin p. Comme p est un plus court chemin, il en est de même de ses sous-chemins et donc $\delta_f(s,u) = \delta_f(s,v)+1$. Par conséquent,

$$\begin{aligned}\delta_f(s,v) &= \delta_f(s,u)-1\\ &\leq \delta_{f'}(s,u)-1\\ &\leq \delta_{f'}(s,v)-2\\ &< \delta_{f'}(s,v),\end{aligned}$$

ce qui mène à une contradiction. C.Q.F.D.

Théorème 1.13. Si l'algorithme de Edmonds-Karp est exécuté sur un réseau de transport G alors le nombre total d'augmentations de flot effectuées par l'algorithme vaut au plus $\mathcal{O}(|S| \times |A|)$.

Preuve. (u,v) dans G_f est dit critique pour un chemin améliorant p si $c_f(p) = c_f(u,v)$.
Quelques propriétés :

1. Tout arc critique disparaît du réseau résiduel après augmentation du flot.
2. Tout chemin améliorant contient au moins un arc critique.

Pour $(u, v) \in A$, nous allons majorer le nombre de fois où (u, v) est un arc critique dans l'algorithme de Edmonds-Karp.

Soit f le flot lorsque pour la première fois (u, v) est critique dans le chemin améliorant.

$$p : s \ldots \rightarrow u \rightarrow v \rightarrow \ldots t \text{ dans } G_f.$$

On a alors $\delta_f(s, v) = \delta_f(s, u) + 1$ et après augmentation, (u, v) disparaît du réseau résiduel.

Pour que (u, v) réapparaisse plus tard il faut que le flot net de u vers v diminue, c'est-à-dire que (v, u) se retrouve sur un chemin améliorant. Soit f' le flot à ce moment-là.

$$p' : t \ldots \leftarrow u \leftarrow v \leftarrow \ldots s \text{ dans } G_{f'}.$$

On a alors $\delta_{f'}(s, u) = \delta_{f'}(s, v) + 1$. Comme d'après le lemme 1.12, $\delta_f(s, v) \leq \delta_{f'}(s, v)$, on peut conclure :

$$\begin{aligned} \delta_{f'}(s, u) &= \delta_{f'}(s, v) + 1 \\ &\geq \delta_f(s, v) + 1 \\ &\geq \delta_f(s, u) + 2. \end{aligned}$$

Comme $\delta_f(s,u)$ est majoré par $|S|$, (u,v) ne peut être un arc critique qu'au plus $\frac{|S|}{2}$ fois. Par conséquent, le nombre total d'augmentations de flot effectuées par l'algorithme vaut au plus $\mathcal{O}(|S| \times |A|)$. C.Q.F.D.

L'algorithme du plus court chemin avec parcours en largeur nécessite un temps de calcul en $\mathcal{O}(|S| + |A|)$ quand le graphe est représenté par des listes d'adjacence.

Exercice 1.6. Exercice 27.2-8 ($|A|$ chemins améliorants) de [CLR94, page 589]. "Comment faire mieux que $\mathcal{O}(|S| \times |A|)$ itérations ?"

1. Montrer que si $|f| > 0$ alors il existe un chemin $s \xrightarrow{*} t$ où tous les flots nets sont strictement positifs (sur le graphe où seuls les flots nets positifs où nuls sont représentés).
2. En déduire qu'on peut toujours trouver un flot maximal dans un réseau $G = (S, A)$ à l'aide d'une séquence d'au plus $|A|$ chemins améliorants.

Exercice 1.7. Problème 27-4 de [CLR94, page 615]. Soit $G = \langle S, A, s, t, c \rangle$ un réseau de transport de capacités entières. Supposons qu'on connaisse un flot maximal de G.

1. Supposons que la capacité d'un seul arc $(u,v) \in A$ augmente de 1. Donner un algorithme en $\mathcal{O}(|S| +$

$|A|$) permettant de mettre à jour le flot maximal.

2. Supposons que la capacité d'un seul arc $(u, v) \in A$ diminue de 1. Donner un algorithme en $\mathcal{O}(|S| + |A|)$ permettant de mettre à jour le flot maximal.

1.4 Couplage maximal dans un graphe biparti

Exemple de problème combinatoire qui peut se ramener à un problème de flot maximal.

Définition 1.12. (couplage, couplage maximal) Soit $G = (S, A)$ un graphe non orienté. Un couplage M est un sous-ensemble de A tel que chaque sommet $v \in S$ a au plus une arête de M qui soit incidente à v. M est un couplage maximal ssi M est un couplage et pour tous les couplages M', $|M'| \leq |M|$.

On s'intéresse dans la suite aux graphes bipartis avec $S = L \cup R$ (L, R une partition de S) et toutes les arêtes de A passent entre L et R.

Problème : calcul d'un couplage maximal pour un graphe biparti G.

À G on associe un réseau de transport G' tel que l'on puisse trouver une correspondance entre flox maximal dans G' et couplage maximal dans G.

Définition 1.13. (réseau de transport correspondant) On construit le réseau de transport $G' = \langle S', A', s, t, c\rangle$ à partir du graphe biparti $G = (S, A)$.

- $S' = \{s, t\} \cup S$ avec $s, t \notin S$;
- $A' = \{(s, u) : u \in L\} \cup \{(u, v) : \{u, v\} \in A, u \in L\} \cup \{(v, t) : v \in R\}$;

– c retourne la valeur 1 pour chaque arc de A'.

Figure 27.9 de [CLR94, page 591].

Lemme 1.14. Soient G et G' définis comme ci-dessus.

(I) Si M est un couplage de G alors il existe un flot à valeurs entières f dans G', avec $|f| = |M|$.

(II) Si f est un flot à valeurs entières de G', alors il existe un couplage M de G de cardinal $|M| = |f|$.

Preuve. (I) Soit M un couplage de G. Le flot f est défini de la façon suivante :

- si $\{u, v\} \in M$, alors $f(s, u) = f(u, v) = f(v, t) = 1$ et $f(u, s) = f(v, u) = f(t, v) = -1$;
- si $\{u, v\} \in A \setminus M$, $f(s, u) = f(u, v) = f(v, t) = f(u, s) = f(v, u) = f(t, v) = 0$.

On peut facilement vérifier que f est un flot (car M est un couplage). Le flot net de la coupe $(L \cup \{s\}, R \cup \{t\})$ est précisemment $|M|$ et donc par le lemme 1.8, $|M| = |f|$.
(II) Soit f un flot à valeurs entières de G'.

$$M = \{\{u, v\} : u \in L,\ v \in R,\ f(u, v) > 0\}.$$

M est un couplage car f vérifie la condition de capacité et chaque capacité de G' vaut 1. Vérifions que $|M| = |f|$. On peut remarquer que pour $u \in L$ couvert, on a $f(s, u) = 1$ et pour $\{u, v\} \in A \setminus M$, on a $f(u, v) = 0$.

$$\begin{aligned} |M| &= f(L, R) \\ &= f(L, S') - f(L, L) - f(L, s) - f(L, t) \\ &= 0 - 0 + f(s, L) - 0 \end{aligned}$$

$= f(s, S')$
$= |f|$.

C.Q.F.D.

Théorème 1.15. (Thèorème de l'intégralité) Si c ne prend que des valeurs entières, le flot maximal f^* produit par la méthode de Ford-Fulkerson vérifie que $|f^*|$ est entier (ainsi que chaque valeur $f[u, v]$).

Corollaire 1.16. Le cardinal d'un couplage maximal dans un graphe biparti G est la valeur d'un flot maximal dans son réseau de transport G' correspondant.

Le cardinal du couplage maximal est au plus $min(|L|, |R|)$ et donc l'algorithme de Ford-Fulkerson a $\mathcal{O}(|S|)$ itérations : problème résoluble en temps $\mathcal{O}(|S| \times |A|)$.

Exercice 1.8. Exercice 27.3-4 (couplage parfait) de [CLR94, page 593].

1.5 Algorithme avec préflots

Algorithme	Itérations	Nbre d'opérations
Ford-Fulkerson	$\|f^*\|$	$\mathcal{O}(\|A\| \times \|f^*\|)$
Edmonds-Karp	$\mathcal{O}(\|S\| \times \|A\|)$	$\mathcal{O}(\|S\| \times \|A\|^2)$
Préflots	$\mathcal{O}(\|S\|^2 \times \|A\|)$	$\mathcal{O}(\|S\|^2 \times \|A\|)$
Élèvation vers l'avant		$\mathcal{O}(\|S\|^3)$
No1 Goldberg et Tarjan		$\mathcal{O}(\|S\| \times \|A\| \times log(\frac{\|S\|^2}{\|A\|}))$

Particularités des algorithmes avec préflots :

– modification locale du réseau (un noeud et ses voisins) ;

– ce n'est qu'à la fin que l'on a un flot et de plus maximal ;
sinon, le tableau $f[\cdot,\cdot]$ est un préflot : symétrie, contraintes de capacité et pour $u \in S \setminus \{s\}$,

$$f(S,u) \geq 0 \ ;$$

– opérations élémentaires sur le réseau résiduel :

1. poussée de flot excédentaire depuis un sommet vers ses voisins ;
2. élèvation d'un sommet.

Quand un préflot f vérifie que pour $u \in S \setminus \{s,t\}$, $f(S,u) = 0$ alors f est un flot.

Définition 1.14. (excédent de flot, débordement) f préflot, $u \in S \setminus \{s,t\}$. Excédent de flot sur u : $e(u) = f(S,u)$.
u déborde si $e(u) > 0$.

Par définition, u ne peut pas déborder si $u \in \{s,t\}$.

A chaque noeud $u \in S$, on associe une hauteur $h(u)$. $h(s) = |S|$ et $h(t) = 0$ restent constants durant le déroulement de l'algorithme mais pour $u \in S \setminus \{s,t\}$, $h(u)$ peut évoluer. Une poussée ne se produit de u vers v que si les conditions suivantes sont réunies :

1. $e(u) > 0$, "u déborde" ($u \notin \{s,t\}$, par définition) ;
2. $c_f(u,v) > 0$ "capacité résiduelle de u vers v strictement positive" ;
3. $h(u) = h(v) + 1$ "u un peu plus haut que v".

Définition 1.15. (fonction de hauteur) G réseau de transport, f préflot. $h : S \to \mathbb{N}$ est une fonction de hauteur si

– $h(s) = |S|$;
– $h(t) = 0$;
– $h(u) \leq h(v) + 1$ si $(u,v) \in A_f$.

Conséquence immédiate : si $h(u) > h(v) + 1$ alors (u,v) n'est pas dans le graphe résiduel.

Si

1. u déborde (cad $f(S,u) > 0$ et $u \notin \{s,t\}$) ;
2. $c_f[u,v] > 0$;
3. $h[u] = h[v] + 1$;

alors on pousse $d_f[u,v] = min(e[u], c_f[u,v])$ unités de flot de u vers v.

POUSSER(u,v)

1. $d_f[u,v] \leftarrow min(e[u], c_f[u,v])$
2. $f[u,v] \leftarrow f[u,v] + d_f[u,v]$
3. $f[v,u] \leftarrow -f[u,v]$
4. $e[u] \leftarrow e[u] - d_f[u,v]$
5. $e[v] \leftarrow e[v] + d_f[u,v]$

Nombre constant d'étapes de calcul.

Poussée saturante si $c_f[u,v] = 0$ après la poussée.
Après une poussée non saturante, u ne déborde plus.
Si $d_f[u,v] = e[u] = c_f[u,v]$ alors on a une poussée saturante et après u ne déborde plus.

Si

1. u déborde ;
2. pour v tel que $c_f[u,v] > 0$, $h[u] \leq h[v]$;

alors on élève u avec $h[u] = 1 + min\{h[v] : (u,v) \in A_f\}$.

ÉLEVER(u)

1. $h[u] \leftarrow 1 + min\{h[v] : (u,v) \in A_f\}$.

Comme u déborde, il existe v tel que $f[v,u] > 0$ et donc $c_f[u,v] = c[u,v] - f[u,v] = c[u,v] + f[v,u] > 0$.

Procédure d'initialisation : chaque arc partant de la source est rempli au maximum.

– $h[s] = |S|$ et pour $u \in S \setminus \{s\}$, $h[u] = 0$;

– $f[u,v] = \begin{cases} c[u,v] \text{ si } u = s \\ -c[u,v] \text{ si } v = s \\ 0 \text{ sinon.} \end{cases}$

– mettre à jour $e[u]$ pour $u \in S$.

Vérifions que $h[.]$ initiale est une fonction de hauteur. $u \rightarrow v$ dans G_f implique $u \neq s$, $h[u] = 0$, et donc $h[u] \leq h[v] + 1$.

Algorithme de préflot générique :
PRÉFLOT GÉNÉRIQUE(G)

1. INITIALISER-PRÉFLOT(G, s) ;
2. TANT QUE il est possible d'appliquer une opération de poussée ou d'élévation
   FAIRE choisir une poussée ou une élèvation applicable, et l'exécuter.
3. RETOURNER f.

Quelques propriétés élémentaires :

1. à tout sommet qui déborde, on peut appliquer soit une élévation soit une poussée (mais pas les deux à la fois), à supposer que $h[.]$ est une fonction de hauteur (cf. le lemme 1.17) ;
2. les hauteurs ne diminuent jamais.

Lemme 1.17. Soit G un réseau de transport. Pendant l'exécution de PRÉFLOT GÉNÉRIQUE(G), l'attribut $h[.]$ est une fonction de hauteur.

Preuve. Récurrence sur le nombre d'opérations élémentaires effectuées.

À l'initialisation, h est une fonction de hauteur.

Montrons que h demeure une fonction de hauteur après l'exécution de ÉLEVER(u). Si $(u, v) \in A_f$ alors on a bien

$h[u] \leq h[v]+1$ car $h[u] = 1 + min\{h[w] : (u,w) \in A_f\}$. Si $(w,u) \in A_f$ et $h[w] \leq h[u]+1$ avant l'exécution de ÉLEVER(u), alors $h[w] \leq h[u]+1$ après car $h[u]$ croît.

Montrons que h demeure une fonction de hauteur après l'exécution de POUSSER(u,v). Si (v,u) est ajouté à A_f (poussée saturante ou pas), alors $h[v] = h[u]-1$ et donc h est une fonction de hauteur (car alors $h[v] \leq h[u]+1$). Si (u,v) est supprimé de A_f (poussée saturante) alors il n'y a plus de contraintes sur (u,v). C.Q.F.D.

Lemme 1.18. Soient G un réseau de transport, f un préflot, et h une fonction de hauteur sur S. Alors, il n'y a pas de chemin entre s et t dans le réseau résiduel G_f.

Preuve. Laissée en exercice. C.Q.F.D.

De même, on peut vérifier que $f[.,.]$ demeure un préflot pendant l'exécution de PRÉFLOT GÉNÉRIQUE(G).

Théorème 1.19. Si PRÉFLOT GÉNÉRIQUE(G) termine alors le préflot calculé est un flot maximal.

Preuve. Si PRÉFLOT GÉNÉRIQUE(G) termine alors aucune opération n'est applicable, c'est-à-dire aucun noeud ne déborde et donc f est un flot. Comme h est une hauteur, il n'y a pas de chemin entre s et t dans G_f. Par le théorème 1.10 (flot maximal et coupe minimale), f est maximal. C.Q.F.D.

Nous allons maintenant montrer que le nombre d'opérations exécutées est borné.

– Nombre d'élèvations $\leq 2 \times |S|^2$;
– Nombre de poussées saturantes $2 \times |S| \times |A|$;
– Nombre de poussées non saturantes $4 \times |S|^2(|S| + |A|)$.

Il y a moyen de remettre une partie du trop plein à la source quand u déborde.

Lemme 1.20. Soient G un réseau de transport et f un préflot. Si u est un sommet débordant, alors il existe un chemin simple de u vers s dans G_f.

Preuve. U : ensemble de noeuds accessibles de u dans G_f. Supposons que $s \notin U$ et posons $\overline{U} = S \setminus U$.

$v \in U$ et $w \in \overline{U}$

$\rightarrow \langle v, w \rangle \notin A_f$ (sinon $w \in U$)
$\rightarrow c(v, w) - f(v, w) = 0$
$\rightarrow f(v, w) \geq 0$
$\rightarrow f(w, v) \leq 0.$

Ainsi $f(\overline{U}, U) \leq 0$.

$$\begin{aligned} e(U) &= f(S, U) \\ &= f(\overline{U}, U) + f(U, U) \\ &= f(\overline{U}, U) \text{ ($f(U, U) = 0$ même si f est un préflot)} \\ &\leq 0. \end{aligned}$$

Comme nous avons l'hypothèse que $U \subseteq S \setminus \{s\}$, pour $v \in U$, $e(v) = 0$. Rappelons $f[.,.]$ est un préflot, cad pour $w \in S \setminus \{s\}$, $f(S, w) \geq 0$. En particulier $e(u) = 0$, une contradiction. C.Q.F.D.

Lemme 1.21. À chaque instant de l'exécution de PRÉFLOT GÉNÉRIQUE(G), $h[u] \leq 2 \times |S| - 1$ pour tous les sommets $u \in S$.

Preuve. Les valeurs constantes $h[s]$ et $h[t]$ vérifient la propriété.

Soient $h_1, \ldots, h_n = h$ les valeurs successives de la fonction de hauteur dans l'exécution de l'algorithme et $u \in S \setminus \{s, t\}$. Par exemple, ces valeurs sont celles obtenues après élèvation. Si u ne déborde pas de h_1 à h_n, alors $h_1(u) = \ldots = h_n(u) = 0$, ce qui vérifie aussi la propriété. Sinon, soit h_l la dernière valeur de la fonction où u déborde. On distingue deux cas.
Cas 1 : $l = n$.
Comme u déborde, d'après le lemme 1.20, il existe un chemin simple de u vers s dans G_f où f est la valeur du préflot à ce moment. Soit $p = \langle v_0, \ldots, v_k \rangle$ avec $v_0 = u$, $v_k = s$ et $k \leq |S| - 1$ (p est simple). Pour $0 \leq i \leq k - 1$, $(v_i, v_{i+1}) \in A_f$ et donc $h[v_i] \leq h[v_{i+1}] + 1$. Ainsi $h[u] \leq h[s] + k \leq 2 \times |S| - 1$.
Cas 2 : $l < n$.

L'opération juste après h_l est une poussée, car une élèvation ne modifie pas l'ensemble des sommets qui débordent. Pour $j \geq l+1$, $h_{l+1}(u) = h_j(u)$. Comme u déborde, d'après le lemme 1.20, il existe un chemin simple de u vers s dans G_f où f est la valeur du préflot à ce moment.

Soit $p = \langle v_0, \ldots, v_k \rangle$ avec $v_0 = u$, $v_k = s$ et $k \leq |S|-1$ (p est simple). Pour $0 \leq i \leq k-1$, $(v_i, v_{i+1}) \in A_f$ et donc $h_l[v_i] \leq h_l[v_{i+1}] + 1$. Ainsi $h_l[u] \leq h_l[s] + k \leq 2 \times |S| - 1$. Par conséquent $h[u] \leq 2 \times |S| - 1$. C.Q.F.D.

Corollaire 1.22. À chaque instant de l'exécution de PRÉFLOT GÉNÉRIQUE(G), le nombre d'élévation par sommet est au plus égal à $2 \times |S| - 1$.

Nombre d'élévation total inférieur à $2 \times |S|^2$ (seuls les noeuds de $S \setminus \{s, t\}$ peuvent être élevés).

Lemme 1.23. À chaque instant de l'exécution de PRÉFLOT GÉNÉRIQUE(G), le nombre de poussées saturantes vaut au plus $2 \times |S| \times |A|$.

Preuve. Pour $u, v \in S$, on considère les poussées de u vers v et aussi celles de v vers u. Si de telles poussées existent, alors soit $(u, v) \in A$, soit $(v, u) \in A$.

Si on effectue une poussée saturante de u vers v, (u, v) disparaît de A_f et pour qu'il y revienne, il faut au préalable effectuer une poussée de v vers u.

Si par exemple, une poussée saturante a eu lieu de u vers v, alors $h[u] = h[v] + 1$ et si on fait ensuite une poussée de v vers u alors $h[v] = h[u] + 1$: entre deux poussées saturantes sur (u, v), $h[v]$ augmente au moins de 2. De même, entre deux poussées saturantes sur (v, u), $h[u]$ augmente au moins de 2.

Soit I la séquence d'entiers $h[u] + h[v]$ entre deux poussées saturantes sur (u, v) ou (v, u). À la première poussée $h[u] + h[v] \geq 1$. À la dernière poussée $h[u] + h[v] \leq 4 \times |S| - 3$ ($= 2 \times (2 \times |S| - 1) - 1$). Comme entre deux poussées saturantes, $h[u] + h[v]$ augmente de 2, le nombre de poussées saturantes entre u et v est bornée par $2 \times |S| - 1$. Pour l'ensemble des arcs, on obtient alors au plus $2 \times |S| \times |A|$ poussées saturantes. C.Q.F.D.

Lemme 1.24. À chaque instant de l'exécution de PRÉFLOT GÉNÉRIQUE(G), le nombre de poussées non saturantes vaut au plus $4 \times |S|^2(|S| + |A|)$.

Preuve. Fonction potentiel (X ens. des sommets débordants) :

$$\Phi = \sum_{v \in X} h[v].$$

1. ÉLEVER(u) fait croître Φ au plus de $2 \times |S|$ (borne supérieure des hauteurs) ;
2. une poussée saturante de u vers v fait croître Φ au plus de $2 \times |S|$ si $e[u] \neq c_f[u, v]$;

3. une poussée saturante de u vers v fait décroître Φ d'au moins 1[1] si $e[u] = c_f[u, v]$: u ne déborde plus et $h[v] + 1 = h[u]$;
4. une poussée non saturante de u vers v fait décroître Φ d'au moins 1 : u ne déborde plus et $h[v]+1 = h[u]$.

(une poussée de u vers v fait toujours déborder v.)

Quantité totale d'augmentations :

$$(2\times|S|)\times 2\times|S|^2+(2\times|S|)\times 2\times|S|\times|A| \;=\; 4\times|S|^2(|S|+|A|).$$

La quantité totale de diminution est bornée par $4\times|S|^2(|S|+|A|)$ et donc il ne peut y avoir plus de $4 \times |S|^2(|S| + |A|)$ poussées non saturantes (la valeur initiale de Φ est 0). C.Q.F.D.

Théorème 1.25. L'exécution de PRÉFLOT GÉNÉRIQUE(G) termine et applique $\mathcal{O}(|S|^2\times|A|)$ opérations élémentaires.

Corollaire 1.26. Il existe une implémentation de l'algorithme de préflot générique qui s'exécute en temps $\mathcal{O}(|S|^2\times|A|)$.

Récapitulatif :

1. $|S| \leq 2 \times |A|$;
2. poussées non saturantes : $4 \times |S|^2(|S| + |A|)$,

[1] exactement de 1 si $h[v] = 0$ ou si v ne déborde pas avant la poussée.

3. poussées saturantes : $2 \times |S| \times |A|$,
4. élèvations : $2 \times |S|^2$.

POUSSER(u, v) exécuté en temps constant & ÉLEVER(u) en $\mathcal{O}(|S|)$. Il faut vérifier que les conditions d'applications de ces opérations n'induisent pas un coût supplémentaire.
Preuve. En exercice. C.Q.F.D.

2 NP-complétude

Quelques rappels juste utiles pour justifier l'intérêt des algorithmes d'approximation.

2.1 Temps polynomial

Problème de décision : $L \subseteq \Sigma^*$ (un langage).

Présentation différente :

entrée : $x \in \Sigma^*$

sortie : 1 si $x \in L$, 0 sinon.

Modèles de calcul :

- Machine de Turing (cf. le cours de calculabilité) ;
- RAM.

Définition 2.1. (DTIME(f)) Soit $f : \mathbb{N} \to \mathbb{N}$ et $L \subseteq \Sigma^*$. L appartient à la classe DTIME(f) s'il existe une machine déterministe et une constante c qui sur toute entrée x de longueur n, résout le problème en au plus $c \times f(n)$ pas de calcul.

$$\mathrm{P} = \bigcup_{k>0} \mathrm{DTIME}(n \mapsto n^k).$$

2.2 Temps polynomial avec machines non déterministes

NTIME(f) défini comme DTIME(f) mais avec des machines non-déterministes.

$$\mathrm{NP} = \bigcup_{k>0} \mathrm{NDTIME}(n \mapsto n^k).$$

Lemme 2.1. P $\subseteq$ NP.

Définition 2.2. (relation polynomiale) Soit Σ un alphabet. Une relation binaire $R \subseteq \Sigma^* \times \Sigma^*$ est une relation polynomiale si pour $x, y \in \Sigma^*$:

(I) Décider si $R(x, y)$ peut être vérifié en temps polynomial en $|x| + |y|$.

(II) Il existe un polynôme p tel que si $R(x, y)$ alors $|y| \leq p(|x|)$.

Une autre caractérisation de NP :

Définition 2.3. (NP) $L \in \mathrm{NP}$ ssi il existe une relation polynomiale R telle que $L = \{x : \text{ il existe un } y \text{ tel que } R(x, y)\}$.

Chaque $x \in L$ a un certificat y vérifiable en temps polynomial.

SAT :

entrée : un ensemble X de clauses propositionnelles

sortie : 1 si X est satisfaisable, 0 sinon.

Si X contient les variables propositionnelles $\mathrm{p}_1, \ldots, \mathrm{p}_n$ et X est satisfaisable, alors le certificat est une valuation $\{\mathrm{p}_1, \ldots, \mathrm{p}_n\} \to \{\mathrm{Vrai}, \mathrm{Faux}\}$, représentable par une suite de n bits.

3-SAT : restriction de SAT aux clauses ayant exactement 3 littéraux.

CYCLE-HAM :

entrée : un graphe non orienté $G = (S, A)$;
(en fait sa représentation sous forme de chaîne) ;

sortie : 1 si G possède un cycle Hamiltonien, 0 sinon.

Un cycle Hamiltonien est un cycle qui passe une seule fois par chaque sommet de G. Si G possède un cycle Hamiltonien, alors le certificat est une séquence de sommets $v_1, \ldots, v_n$ telle que

- $\{v_i, v_{i+1}\} \in A$ pour $i = 1, \ldots, n-1$,
- $\{v_n, v_1\} \in A$,
- $S = \{v_1, \ldots, v_n\}$ et les v_i sont distincts.

Lemme 2.2. SAT, 3-SAT, et CYCLE-HAM sont dans NP.

Exercice 2.1. Montrer l'équivalence des définitions de NP.

Définition 2.4. (réduction polynomiale) Soient $L \subseteq \Sigma^*$ et $L' \subseteq \Sigma'^*$. L se réduit polynomialement à L' s'il existe $f : \Sigma^* \to \Sigma'^*$ telle que

- $x \in L$ ssi $f(x) \in L'$ pour $x \in \Sigma^*$,
- f est calculable par une machine de Turing en temps polynomial.

Définition 2.5. L est NP-complet si L est dans NP et tout problème de NP peut se réduire polynomialement à L.

La NP-dureté signifie que L est au moins aussi dur que n'importe quel problème de NP.

Théorème 2.3. SAT, 3-SAT, et CYCLE-HAM sont NP-complets.

On ne connait pas de problème NP-dur pouvant être résolu par un algorithme déterministe en temps polynomial (peu probable que cela existe).

2.3 Problèmes NP-complets

Méthode pour démontrer qu'un langage est NP-complet :

1. Montrer que $L \in$ NP.
2. Choisir un langage L' NP-complet connu (la NP-dureté suffit en fait).
3. Trouver une fonction $f : \Sigma'^* \to \Sigma^*$.

4. Montrer que $x \in L'$ ssi $f(x) \in L$.
5. Démontrer que f se calcule en temps polynomial (espace logarithmique).

2.3.1 Le problème de la clique

CLIQUE :

entrée : un graphe non orienté G et $k \geq 0$ (unaire)

sortie : 1 si G a une clique de taille k, 0 sinon.

CLIQUE comme un langage :

$$\text{CLIQUE} = \{\langle G, k\rangle : G \text{ contient une clique de taille } k\}.$$

Théorème 2.4. Le problème de la clique est NP-complet.

Preuve. CLIQUE est dans NP en prenant comme certificat la clique de cardinal $min(k, |S|)$.
Pour montrer que CLIQUE est NP-difficile, on réduit 3-SAT à CLIQUE en temps polynomial. Soit $X = C_1 \wedge \cdots \wedge C_k$ une instance de 3-SAT (chaque C_i est une clause de la forme $l_1^i \vee l_2^i \vee l_3^i$). On construit l'instance suivante $\langle G, k\rangle$ de CLIQUE :

– $G = (S, A)$ avec S l'ensemble

$$\{\langle l_j^i, i\rangle : 1 \leq i \leq k,\ 1 \leq j \leq 3\}.$$

– $\{l_j^i, l_{j'}^{i'}\} \in A$ ssi $i \neq i'$ et $l_j^i \neq \overline{l_{j'}^{i'}}$.

[CLR94, Figure 36.12].

G peut facilement être construit en temps polynomial en $|X|$.

On peut monter que $X = C_1 \wedge \ldots \wedge C_k$ est satisfaisable ssi G a une clique de taille k. C.Q.F.D.

Dans la suite, seule la NP-dureté est intéressante.

Exercice 2.2. Montrer que CLIQUE demeure dans NP si k est codé en binaire.

2.3.2 Le problème de la couverture de sommets

Couverture de sommets :

entrée : un graphe non orienté $G = (S, A)$, un entier k ;

sortie : 1 s'il existe $S' \subseteq S$ tel que $|S'| = k$ et pour $\{u, v\} \in A$, $\{u, v\} \cap S' \neq \emptyset$, 0 sinon.

Théorème 2.5. Le problème de la couverture de sommets est NP-complet.

Preuve. Pour montrer que la couverture de sommets est NP-difficile, on réduit CLIQUE à ce problème en temps polynomial. Soit $\langle G, k \rangle$ une instance de CLIQUE (G graphe non orienté). On construit l'instance suivante $\langle G', k' \rangle$ de la couverture de sommets :

– $G' = (S, A')$ graphe complément de G, cad $A' = \{\{u, v\} : \{u, v\} \notin A,\ u \neq v\}$;

– $k' = |S| - k$.

$\langle G', k' \rangle$ peut se calculer en temps polynomial en la taille de $\langle G, k \rangle$.

Montrons à présent que G a une clique de taille k ssi G' a une couverture de sommets de taille k' (laissé en exercice). C.Q.F.D.

2.4 Le problème du voyageur de commerce

PVC :

entrée : un graphe complet $G = (S, A)$, $c : A \to \mathbb{N}$, k ;

sortie : 1 si G possède une tournée de coût au plus égal à k, 0 sinon.

Tournée : cycle hamiltonien.

Théorème 2.6. Le problème du voyageur de commerce est NP-complet.

Seule la NP-dureté est intéressante.

Preuve. Pour montrer que PVC est NP-difficile, on réduit CYCLE-HAM à PVC en temps polynomial. Soit $G = (S, A)$ une instance de CYCLE-HAM (G graphe non orienté). On construit l'instance suivante de PVC :

– $G' = (S', A')$ avec $S' = S$ et $A' = \{\{u, v\} : u \neq v \in S'\}$
(graphe complet) ;

– $c : A' \to \mathbb{N}$ avec
  $c(u, v) = 1$ si $\{i, j\} \notin A$, $c(u, v) = 0$ sinon.

On peut montrer que G' a une tournée de coût au plus égal à 0 ssi G possède un cycle Hamiltonien. De plus, $\langle G', c', 0\rangle$ peut se calculer en temps polynomial en la taille de G. C.Q.F.D.

2.5 Traiter la NP-complétude

On ne connaît pas d'algorithmes déterministes non exponentiels en temps pour les problèmes NP-complets.

2.5.1 Sous-problèmes

Restreindre la classe d'instances :
– pour ne s'intéresser qu'aux instances rencontrées en pratique par exemple,
– pour déterminer les limites de la NP-dureté.

Lemme 2.7. 2-SAT est dans P.

Preuve. Soit $X = \{l_1^1 \vee l_2^1, \ldots, l_1^n \vee l_2^n\}$ un ensemble de 2-clauses. On construit le graphe orienté $G = (S, A)$ suivant :
– S est l'ensemble des littéraux construits à partir des variables propositionnelles de X.
– si $l \vee l' \in X$ alors $(\neg l, l')$ et $(\neg l', l)$ sont dans A.

On peut montrer que X n'est pas satisfaisable ssi il existe un circuit contenant un littéral et son complément (laissé en exercice). C.Q.F.D.

Par contre, la restriction de PVC avec des coûts dans $\{0, 1\}$ demeure NP-difficile.

2.5.2 Algorithmes pseudo-polynomiaux

Définition 2.6. (algorithme pseudo-polynomial) Un algorithme pour un problème est appelé pseudo-polynomial si son temps d'exécution est polynomial dans la taille de l'entrée et dans la taille de la représentation unaire du plus grand nombre qui apparaît dans l'entrée.

Somme de sous-ensemble :

entrée : un ensemble fini d'entiers E et un but $t \in \mathbb{N}$;

sortie : 1 s'il existe $E' \subseteq E$ tel que $(\Sigma_{x \in E'} x) = t$, 0 sinon.

Théorème 2.8. Le problème "Somme de sous-ensemble" est NP-complet et admet un algorithme pseudo-polynomial.

L'algorithme pseudo-polynomial utilise les concepts de la programmation dynamique.

Définition 2.7. (NP-complet au sens fort) Un problème est NP-complet au sens fort, s'il reste NP-complet même si le plus grand nombre qui apparaît dans l'instance est polynomial par rapport à la taille de l'entrée.

Programmation entière :

entrée : une matrice $k \times n$ notée A, un vecteur de dimension k noté b (à coefficients dans $\mathbb{Z}$) ;

sortie : 1, s'il existe des entiers positifs $x_1, \ldots, x_n$ tels que $Ax \geq b$, 0 sinon.

Théorème 2.9. Le problème "Programmation entière" est NP-complet au sens fort.

La preuve de NP-dureté réduit 3-SAT à Programmation entière et seuls les coefficients 1, 0, et -1 sont utilisés.

Idem pour PVC.

2.5.3 Méthodes probabilistes

Algorithmes Monte-Carlo :
- temps d'exécution polynomial dans tous les cas ;
- un résultat faux peut être produit avec une probabilité strictement inférieure à 1.

En itérant ce type d'algorithmes, on peut se ramener à une probabilité aussi proche de 0 que l'on veut (cf. la classe de complexité **RP**).

Algorithmes de type Las-Vegas :
– temps d'exécution polynomial en moyenne ;
– réponse exacte.
(cf. la classe de complexité **ZPP** (= **RP** $\cap$ co-**RP**)). Complexité en moyenne plutôt que complexité dans le pire des cas (mais les preuves sont plus dures que dans le pire des cas).

Du point de vue de la complexité, on considère des machines de Turing non déterministes dont les modes d'acceptation sont différents des classes non probabilistes. Introduction de nouvelles classes de complexité : **RP**, **ZPP**, **PP**, **BPP**, ...

2.5.4 Complexité paramétrée

Couverture de sommets résoluble en temps $\mathcal{O}(p(n)^k)$ où n est la taille de l'entrée et $p(.)$ est un polynôme.

Ensemble dominant :

entrée : un graphe non orienté $G = (S, A)$, un entier k ;

sortie : 1 s'il existe $S' \subseteq S$ tel que $|S'| = k$ et pour $u \in S$, soit $u \in S'$, soit u a un voisin dans S', 0 sinon.

Théorème 2.10. Ensemble dominant est un problème NP-complet.

Ensemble dominant résoluble en temps $\mathcal{O}(p(n)^k)$ où n est la taille de l'entrée et $p(.)$ est un polynôme.

Peut-on éviter d'avoir k en exposant ?

Par exemple, existe-t-il un algorithme en temps $\mathcal{O}(f(k)\times p(n))$ où $p(.)$ est un polynôme ?

Amélioration sensible en pratique si k varie peu (ou ne prend que quelques valeurs).

Théorème 2.11. Il existe un algorithme pour couverture de sommets en temps $\mathcal{O}((\frac{53}{40})^k \times k^2 + (k \times n))$.

Il est peu probable qu'il existe un algorithme en $\mathcal{O}(f(k)\times p(n))$ pour le problème de l'ensemble dominant, d'après un résultat de la théorie de la complexité paramétrée (la "W[2]-complétude" de la version paramétrée du problème).

Complexité paramétrée : branche de la théorie de la complexité qui distingue des paramètres dans les ins-

tances d'un problème et qui analyse les coûts algorithmes en tenant compte de cette distinction.

2.5.5 Algorithmes d'approximations

Définition 2.8. (problème d'optimisation) Un problème d'optimisation est une structure $P = \langle L_1, L_2, R, val, but \rangle$ tel que :

- $L_1 \subseteq \Sigma^*$, ensemble d'instances ;
- $L_2 \subseteq \Sigma^*$, ensemble de solutions ;
- $R \subseteq L_1 \times L_2$ (relier les solutions aux instances), $(x, y) \in R$: y est une solution de x ;
- $val : R \rightarrow D$ ($D = \mathbb{N}, \mathbb{Q}, \mathbb{R}, \ldots$) ;
- $but \in \{max, min\}$.

Exemple 2.12. CLIQUE :

- L_1 : codages des graphes non orientés ;
- L_2 : codages des ensembles de sommets dans un graphe ;
- R : cliques de graphes ;
- $val : R \rightarrow \mathbb{N}$, cardinalité de la clique ;
- $but = max$.

Terminologie :

- max : objectif.

– min : coût.

Étant donné un problème d'optimisation P, on peut définir plusieurs problèmes :

– Problème de décision :

entrée : $x \in \Sigma^*$, $k \in D$;

sortie : 1 s'il existe $(x, y) \in R$ avec $val(x, y) \geq k$ si $but = max$ [resp. avec $val(x, y) \leq k$ si $but = min$], 0 sinon.

– Problème de recherche :

entrée : $x \in \Sigma^*$, $k \in D$;

sortie : y s'il existe $(x, y) \in R$ avec $val(x, y) \geq k$ si $but = max$ [resp. avec $val(x, y) \leq k$ si $but = min$], 0 sinon.

– Problème d'optimisation :

entrée : $x \in \Sigma^*$;

sortie : $but\{val(x, y) : (x, y) \in R\}$.

– Problème d'optimisation constructif :

entrée : $x \in \Sigma^*$;

sortie : y' tel que $(x, y') \in R$ et $val(x, y') = but\{val(x, y) : (x, y) \in R\}$.

Les problèmes sont par ordre croissant de difficulté, en particulier si $val(x, y)$ se calcule en temps polynomial en $|x| + |y|$.

Lemme 2.13. Soit $P = \langle L_1, L_2, R, val, but \rangle$ tel que $D = \mathbb{N}$, et $but\{val(x, y) : (x, y) \in R\} \leq 2^{p(|x|)}$ pour tout x, où $p(.)$ est un polynôme. Le problème d'optimisation peut être résolu en temps polynomial ssi le problème de décision peut être résolu en temps polynomial.

Preuve. De façon immédiate, si le problème d'optimisation peut être résolu en temps polynomial alors le problème de décision peut être résolu en temps polynomial. Un seul appel à l'algorithme du problème d'optimisation résoud le problème de décision.

Réciproquement, en utilisant une simple dichotomie, le problème d'optimisation peut être résolu avec $p(|x|)$ appels à l'algorithme du problème de décision. C.Q.F.D.

3 Algorithmes d'approximation

On considère des problèmes d'optimisation constructifs de la forme :

entrée : $x \in \Sigma^*$;

sortie : y' tel que $(x, y') \in R$ et $val(x, y') = but\{val(x, y) : (x, y) \in R\}$.

où toutes les valeurs sont dans $\mathbb{N}$.

3.1 Bornes de performance pour les algorithmes d'approximation

Définition 3.1. (borne $\rho(n)$) Un algorithme pour un problème d'optimisation constructif a une borne $\rho(n)$ si pour toute entrée x de taille n, la solution produite y vérifie

$$max(\frac{C}{C^*}, \frac{C^*}{C}) \leq \rho(n),$$

où $C = val(x, y)$ et $C^* = but\{val(x, y') : (x, y') \in R\}$.

Les deux rapports dans la définition permettent de traiter les cas $but = min$ et $but = max$.

Il peut être quelquefois plus commode de travailler avec une erreur relative.

Définition 3.2. (erreur relative $\epsilon(n)$) Un algorithme pour un problème d'optimisation constructif a une erreur relative $\epsilon(n)$ si pour toute entrée x de taille n, la solution

produite y vérifie

$$\frac{|C - C^*|}{max(C, C^*)} \leq \epsilon(n),$$

où $C = val(x, y)$ et $C^* = but\{val(x, y') : (x, y') \in R\}$.

Le maximum dans le dénominateur permet de traiter les cas $but = min$ et $but = max$.

Lemme 3.1. Si on pose $\epsilon(n) = max_{|x|=n} \frac{|C-C^*|}{C^*}$ et $\rho(n) = max_{|x|=n} max(\frac{C}{C^*}, \frac{C^*}{C})$, alors $\epsilon(n) \leq \rho(n) - 1$.

Définition 3.3. (schéma d'approximation polynomial) Un problème d'optimisation P a un schéma d'approximation polynomial si pour toute constante $\epsilon > 0$, il existe un algorithme (en temps) polynomial pour P ayant une erreur relative ϵ.

On peut espérer encore un peu mieux :

Définition 3.4. (schéma d'approximation entièrement polynomial) Un problème d'optimisation P a un schéma d'approximation entièrement polynomial si pour toute constante $\epsilon > 0$, il existe un algorithme pour P polynomial en $\frac{1}{\epsilon}$ et en la taille de l'entrée ayant une erreur relative ϵ.

Le temps d'exécution peut être par exemple de l'ordre de $\frac{1}{\epsilon^3} \times n^2$.

3.2 Techniques de conception des algorithmes d'approximation

- méthode gloutonne (heuristique simple mais pas de très bonne qualité),
- programmation dynamique,
- programmation linéaire (on arrondit les solutions),
- recherche locale,
- algorithmes probabilistes.

3.3 Le problème de la couverture de sommet

Le problème d'optimisation constructif associé au problème de couverture de sommets consiste à rechercher une couverture de sommet ayant un nombre mimimal de sommets.

COUVERTURE-SOMMET-APPROCHÉE(G)

1. $C \leftarrow \emptyset$;
   (C : couverture courante)
2. $A' \leftarrow A[G]$;
3. TANT QUE $A' \neq \emptyset$
   FAIRE
   Soit $\{u, v\}$ une arête arbitraire de A'.
   $C \leftarrow C \cup \{u, v\}$.
   Supprimer de A' toutes les arêtes incidentes soit à u soit à v.

4. RETOURNER C.

Temps d'exécution en $\mathcal{O}(|A|)$ (avec une structure de données adéquate) et COUVERTURE-SOMMET-APPROCHÉE(G) retourne une couverture de sommets de G.

Théorème 3.2. COUVERTURE-SOMMET-APPROCHÉE possède une borne de 2.

Preuve. Soit E l'ensemble courant des arêtes choisies dans A' ayant donné lieu à une augmentation de C.
Deux arêtes quelconques de E n'ont pas de sommets communs.
La valeur courante de C vérifie $|C| = 2 \times |E|$. En particulier $|C_{fin}| = 2 \times |E_{fin}|$.
Toute couverture de G, et en particulier une couverture minimale C^*, doit contenir au moins un sommet par arête de E_{fin}. Ainsi $|C^*| \geq |E_{fin}|$.
Par conséquent le cardinal de COUVERTURE-SOMMET-APPROCHÉE(G) est au plus $2 \times |C^*|$. C.Q.F.D.

3.4 Le problème du voyageur de commerce

3.4.1 Le problème avec inégalité triangulaire

Définition 3.5. (coût) Soit $G = (S, A)$ un graphe non orienté complet muni d'une fonction de coût $c : A \to \mathbb{N}$. Pour $E = e_1 \cdots e_n \in A^*$, $c(E) = \Sigma c(e_i)$.

Inégalité triangulaire : pour $u, v, w \in S$,

$$c(u, w) \leq c(u, v) + c(v, w).$$

Condition naturelle pour de nombreux graphes. On s'intéresse ici aux instances de PVC vérifiant l'inégalité triangulaire.

Quelques rappels :

1. un arbre couvrant T pour $G = (S, A)$ graphe connexe non orienté est un sous-ensemble de A formant un arbre couvrant tous les sommets de S.
2. algorithme de Prim calculant un arbre couvrant de poids minimal en temps $\mathcal{O}(|S|^2)$ avec le graphe représenté par une matrice d'adjacence.

TOURNÉE-PVC-APPROCHÉE(G, c)

1. choisir un sommet $r \in S[G]$ (la racine) ;

2. construire un arbre couvrant de poids minimal T pour G à partir de la racine r (à l'aide de PRIM-ACM(G, c, r)) ;
3. soit L la liste des sommets visités lors d'un parcours préfixe de T ;
4. RETOURNER le cycle hamiltonien H qui visite les sommets dans l'ordre de L.

Temps de calcul en $\mathcal{O}(|S|^2)$.
Dérouler l'algorithme sur [CLR94, Figure 37.2].

Théorème 3.3. TOURNÉE-PVC-APPROCHÉE possède une borne de 2 pour les instances du problème du voyageur de commerce vérifiant l'inégalité triangulaire.

Preuve. Soit H^* une tournée optimale et H la tournée retournée par TOURNÉE-PVC-APPROCHÉE(G, c). Nous allons montrer que $c(H) \leq 2 \times c(H^*)$.

En enlevant une arête à H^*, on obtient un arbre couvrant T^* de poids inférieur à $c(H^*)$ (puisque l'on a enlevé une arête). Donc pour tout arbre couvrant de poids minimal T' pour G,

$$c(T') \leq c(T^*) \leq c(H^*).$$

En particulier $c(T) \leq c(H^*)$ pour l'arbre calculé dans TOURNÉE-PVC-APPROCHÉE(G, c).

Soit PP un parcours en profondeur de T qui liste les sommets dès qu'ils sont visités pour la première fois et également quand ils sont à nouveau traversés après la visite d'un sous-arbre.

PP visite exactement 2 fois chaque arête et donc $c(PP) = 2 \times c(T)$. Ainsi $c(PP) \leq 2 \times c(H^*)$.

Cependant PP n'est pas une tournée. Soit H la tournée obtenue à partir de PP qui consiste à ne garder que la première occurrence de chaque sommet. Cela correspond à un parcours préfixe de T. Comme l'inégalité triangulaire est vérifiée, $c(H) \leq c(PP)$.

Par conséquent, $c(H) \leq 2 \times c(H^*)$. C.Q.F.D.

3.4.2 Le problème général

Dans le cas général (sans la condition de l'inégalité triangulaire), l'existence d'une borne est improbable (à moins que P = NP).

Théorème 3.4. Si P $\neq$ NP et $\rho \geq 1$, il n'existe aucun algorithme d'approximation en temps polynomial de borne ρ pour le problème général PVC.

Preuve. Supposons par l'absurde qu'un algorithme d'approximation AA pour PVC en temps polynomial existe pour un $\rho \geq 1$. Sans perte de généralité, supposons ρ entier (cela permet de définir c avec des valeurs entières).

Nous allons montrer comment résoudre CYCLE-HAM en temps polynomial en utilisant AA. Comme CYCLE-HAM est NP-complet, alors cela induit que P = NP.

Soit $G = (S, A)$ une instance de CYCLE-HAM (G graphe non orienté). Soit $G' = (S, A')$ et $c : S \times S \rightarrow \mathbb{N}$ une instance de PVC définie de la façon suivante :

- $A' = \{\{u, v\} : u, v \in S, u \neq v\}$ (complétude de G') ;
- $c(u, v) = 1$ si $\{u, v\} \in A$, $c(u, v) = \rho \times |S| + 1$ sinon.

G' et c peuvent être facilement obtenus en temps polynomial en $|G|$.

Si G possède un chemin Hamiltonien, alors G' a une tournée de coût $|S|$.

Si G ne possède pas de chemin Hamiltonien, alors une tournée de G' doit utiliser une arête qui n'est pas dans A et donc le coût de cette tournée est au moins

$$(\rho \times |S| + 1) + (|S| - 1) > \rho \times |S|.$$

Ainsi G possède un chemin Hamiltonien ssi G' a une tournée de coût $|S|$.

Si G possède un chemin Hamiltonien, alors AA retourne obligatoirement la tournée de coût $|S|$ grâce à la borne ρ et à l'écart entre $|S|$ et $\rho \times |S| + 1$. Si G ne possède pas de chemin Hamiltonien, alors AA retourne une tournée de coût supérieure à $\rho \times |S|$. Ainsi AA décide CYCLE-HAM en temps polynomial. C.Q.F.D.

Exercice 3.1. [CLR94, Exercice 37.2-1]. NP-complétude avec inégalité triangulaire.

Exercice 3.2. [CLR94, Exercice 37.2-2]. Heuristique du point le plus proche.

Références

[CLR94] Th. Cormen, Ch. Leiserson, and R. Rivest. *Introduction à l'algorithmique*. Dunod, 1994.

[Gar00] B. Garcia. Recherche opérationnelle, 2000. Notes de cours accessibles sur le web.