Optimisation Combinatoire

Chien-Chen Huang

19 novembre 2024

Table des matières

1 Max-Flow (min-cut)

1.1 Ford-Fulkerson

Définition 1.1 Soit $G = (V, E)$ un graphe orienté, $c : E \to \mathbb{R}^+$ une fonction de capacité et $s, t \in V$ deux sommets terminaux, $f : E \to \mathbb{R}^+$ est un flot si :

1. $0 \leq f(e) \leq c(e), \forall e \in E$

2. $\sum_{e\in\delta^-(v)} f(e) = \sum_{e\in\delta^+(v)} f(e), \forall v \in V \setminus \{s,t\}$ (Conservation du Flot).

On va s'intéresser au problème suivant :

MAX-FLOW	
Entrée:	$G=(V,E)$ un graphe orienté, $c : E \to \mathbb{R}^+$ une fonction de capacité et $s,t \in V$ deux sommets terminaux
Sortie:	Une fonction de flot f maximale, c'est à dire avec un volume maximal : $\sum_{e\in\delta^+(s)} f(e) - \sum_{e\in\delta^-(s)} f(e)$

Théorème 1.1 On rappelle qu'obtenir un flot maximal est équivalent à obtenir une coupe de poids minimal.

On définit pour cela le réseau résiduel d'une fonction de flot f :

Définition 1.2 Étant donné un flot f, pour chaque arrête $e=(u,v) \in E$, on définit :

$$(u,v) \in F \text{ si } c(e) - f(e) > 0$$
$$(v,u) \in F \text{ si } f(e) > 0$$

Dans le premier cas, on définit $u(e) = c(e) - f(e)$. Dans le deuxième cas on définit $u(e) = f(e)$.

Proposition 1.1 Si on pousse de u à v, i.e. si $f(((u,v)) > 0$, alors (v,u) doit apparaître dans le réseau résiduel.

On propose alors l'algorithme de Ford-Fulkerson, se basant sur des chemins augmentants pour résoudre le problème :

Algorithme 1 Ford-Fulkerson

Tant que $G(f)$ a un chemin de s à t noté p, on pousse le plus possible le long du chemin p.

Théorème 1.2 Si l'algorithme de Ford-Fulkerson termine, alors f est un flot maximal.

Démonstration. Soit $U \subseteq V$ l'ensemble des sommets atteignables depuis s dans $G(f)$. On a par 1.1

$$\begin{aligned}\sum_{e\in\delta^+(s)} f(e) - \sum_{e\in\delta^-(s)} f(e) &= \sum_{e\in\delta^+(U)} f(e) - \sum_{e\in\delta^-(U)} f(e) \\ &= \sum_{e\in\delta^+(U)} c(e) = \text{cut size de U}\end{aligned}$$

■

Toutefois la complexité de cet algorithme dépend de la valeur du flot maximum, et celui-ci ne termine même pas pour des réels.

1.2 Push-Relabel

On va chercher un algorithme dont la complexité n'en dépend pas (dit fortement polynomial), ce qui nous amène à la notion de pré-flot :

Définition 1.3 Soit $G=(V,E)$ un graphe orienté, $c : E \to \mathbb{R}^+$ une fonction de capacité et $s,t \in V$ deux sommets terminaux, $f : E \to \mathbb{R}^+$ est un pré-flot si :

1. $0 \leq f(e) \leq c(e), \forall e \in E$
2. $\text{exces}(v) = \sum_{e \in \delta^-(v)} f(e) \geq \sum_{e \in \delta^+(v)} f(e), \forall v \in V \setminus \{s,t\}$.

On va essayer de construire un algorithme dont le principe est cette fois ci d'avancer

Définition 1.4
- Un sommet $v \in V \setminus \{s,t\}$ est dite *actif* si $\text{exces}(v) > 0$.
- Un étiquetage des sommets $d : V \to \mathbb{N}$ est valide si $\forall (u,v) \in G(f)$ (pour f un pré-flot), on a : $d(u) \leq d(v) + 1$.
- Une arête $(u,v) \in G(f)$ est admissible si $d(u) = d(v) + 1$.

On obtient alors l'algorithme suivant, proposé originellement par Andrew Goldberg en 1989 :

Algorithme 2 Push-Relabel

Initialisation : On pose $\forall e \in \delta^+(s), f(e) = c(e)$, sinon $f(e) = 0$. On pose $d(s) = n, d(v) = 0 \forall v \neq s$.

Boucle Tant qu'il existe un sommet actif v, on effectue deux actions :

`Push` S'il existe $(u,v) \in G(f)$ admissible, on pousse $\min(\text{exces}(u), u(e))$ selon l'arête (u,v).

`Relabel` On pose $d(u) = \min_{v|(u,v)\in G(f)} d(v) + 1$

On va donc prouver la correction de cet algorithme. Pour cela, on se base sur les deux lemmes suivants :

Lemme 1.1 Si v est actif, alors v a un chemin orienté vers s dans $G(f)$.

Démonstration. Soit $X \subseteq V$ l'ensemble des sommets ayant un chemin vers s dans $G(f)$. Par l'absurde, il existe $w \in V \setminus X$ actif. On a alors :

$$0 < \sum_{v \in V \setminus X} \left(\sum_{e \in \delta^-(v)} f(e) - \sum_{e \in \delta^+(v)} f(e) \right) = \sum_{e \in \delta^-(v \setminus x)} f(e) - \sum_{e \in \delta^+(V \setminus X)} f(e)$$

Or, $\sum_{e \in \delta^-(v\setminus x)} f(e) = 0$, d'où le résultat. ■

Lemme 1.2 Étant donné un chemin P de u à v, alors, $|P| \geq d(u) - d(v)$, si d est valide.

Démonstration. Si $P = v_0 v_1 \cdots v_x$, puisque d est valide : $d(v_i) \leq d(v_{i+1}) + 1$ pour tout i. D'où, $d(v_0) \leq d(v_x) + x$. D'où le résultat. ■

On obtient un corollaire très utile :

Corollaire 1.1 Pour tout n, $d(u) \leq 2n - 1$.

Démonstration. Si u est actif et $d(u) = 2n$, tout chemin de u à s est de longueur au moins n, ce qui est impossible puisque $|V| = n$. ■

Théorème 1.3 — Correction de Push-Relabel Quand l'algorithme 2 s'arrête, on obtient un max-flow.

Démonstration. Il n'y a jamais de chemin de s à t dans $G(f)$ par 1.2. Par ailleurs, il n'y a pas de sommet actif à l'arrêt de l'algorithme, ce qui signifie qu'on a bien un véritable flot. La preuve de correction de 1 s'applique donc. ■

Théorème 1.4 — Complexité de Push-Relabel L'algorithme 2 s'arrête en temps $\mathcal{O}(V^2 E)$.

Démonstration. On a toujours 3 opérations :

- Le `Relabel` qui prend un temps $\mathcal{O}(V^2)$ (au plus $(n-2) \times (2n-1)$ opérations).

- Le `Push Saturant` (push qui permet à $f((u,v))$ d'atteindre $c((u,v))$). Celui-ci va supprimer l'arête (u,v) de $G(f)$. Pour que l'arc soit réinséré dans $G(f)$ pour un autre push saturant, v doit d'abord être réétiqueté. Ensuite, après un push sur (v,u), u doit être réétiqueté. Au cours du processus, $d(u)$ augmente d'au moins 2 Il y a donc $\mathcal{O}(V)$ push saturants sur (u,v) et donc $\mathcal{O}(VE)$ push saturants au total.
- Le `Push Non-Saturant` qu'on effectue un nombre $\mathcal{O}(V^2E)$ de fois. En effet, borner le nombre de push non-saturants peut se faire à partir d'un argument de potentiel. On utilise la fonction de potentiel $\Phi = \sum_{v \text{ actif}} d(v)$. Il est clair que $\Phi = 0$ à l'initialisation et reste toujours positive durant l'exécution. Par ailleurs, un push non-saturant diminue Φ d'au moins 1. De plus, le relabel et le push augmentent Φ d'au plus 1 et d'au plus $(2V-1)$ respectivement. On a donc : $\Phi \leq (2V-1)(V-2) + (2V-1)(2VE)$. On a donc : $\Phi \leq \mathcal{O}\left(V^2E\right)$.

L'algorithme prend donc un temps $\mathcal{O}\left(V^2E\right)$. ■

Pour essayer d'améliorer l'algorithme on propose la version suivante : Il est clair que l'algorithme reste correcte,

Algorithme 3 Push-Relabel +

Boucle : On choisit le sommet actif v avec la plus haute étiquette, on effectue deux actions :

`Push` S'il existe $(u,v) \in G(f)$ admissible, on pousse $\min(\text{exces}(u), u(e))$ selon l'arête (u,v).

`Relabel` On pose $d(u) = \min_{v|(u,v)\in G(f)} d(v) + 1$

toutefois, on change la complexité pour de V^2E à V^3. On peut même encore améliorer la complexité pour obtenir $\mathcal{O}(V^2\sqrt{E})$, et même $\mathcal{O}(V^{1+o(1)}\log(E))$

1.3 Edmonds-Karp

Algorithme 4 Edmonds-Karp

Pour cet algorithme, on applique Ford-Fulkerson en choisissant le plus court des chemins de s à t.

Théorème 1.5 — Complexité d'Edmonds-Karp L'algorithme de Edmonds-Karp prend un temps $\mathcal{O}(VE^2)$.

Dans la suite, on note $f_0, f_1, \cdots$ les flots obtenus de sorte que f_{i+1} est obtenu du plus court chemin P_i dans $G(f_i)$.

Lemme 1.3 $\forall i$:

- $|P_i| \leq |P_{i+1}|$
- Si P_i et P_{i+1} utilisent deux arcs opposés (i.e. (u,v) et (v,u)), alors $|P_i| + 2 \leq |P_{i+1}|$.

Démonstration. On pose $H = P_i \cup P_{i+1}$ où les arcs opposés sont supprimés. On ajoute alors 2 arcs supplémentaires de t à s. Comme alors H est eulérien, il existe deux chemins disjoints q_1, q_2 de s à t dans H. Notons que toutes les arêtes de H (sauf les arêtes de t à s) sont dans $G(f_i)$. On a de plus $|P_i| \leq q_1, q_2$ d'où

$$2|P_i| \leq |q_1| + |q_2| \leq |H| \leq |P_i| + |P_{i+1}| - 2$$

■

Lemme 1.4 Soit $l < k$ tel que P_l et P_k utilisent des arcs opposés. Alors, $|P_l| + 2 \leq |P_k|$.

Démonstration. La preuve précédente peut être aisément adaptée : On peut supposer que pour $l < i < k$, P_i n'a pas d'arcs opposés avec P_k, sinon le résultat se déduit par récurrence par le lemme précédent. On pose $H = P_l \cup P_k$ où les arcs opposés sont supprimés. On ajoute alors 2 arcs supplémentaires de t à s. Comme alors H est eulérien, il existe

deux chemins disjoints q_1, q_2 de s à t dans H. Notons que toutes les arêtes de H (sauf les arêtes de t à s) sont dans $G(f_l)$. On a de plus $|P_l| \leq q_1, q_2$ d'où

$$2\,|P_l| \leq |q_1| + |q_2| \leq |H| \leq |P_l| + |P_k| - 2$$

■

Lemme 1.5 Un arc dans $G(f)$ peut être une arête bottleneck (c'est à dire une arête avec le moins de capacité) au plus $\mathcal{O}(n)$ fois.

Démonstration. Pour qu'une arête e soit bottleneck une nouvelle fois, il faut que la longueur du nouveau plus court chemin ait augmenté au moins de 2. ■

Démonstration de la Complexité d'Edmonds-Karp. Chaque arête peut être utilisée au plus $\mathcal{O}(n)$, il y a donc au plus nm itérations, et les itérations se font en temps $\mathcal{O}(m)$, ce qui est le résultat. ■

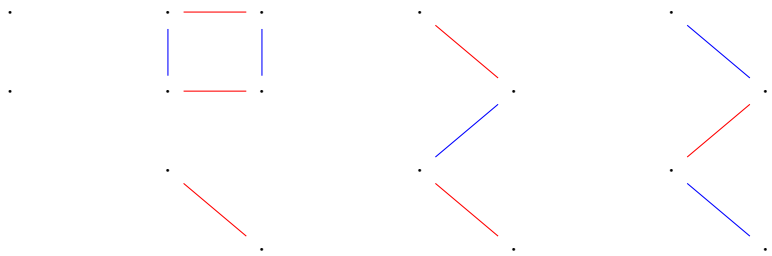
2 Couplage Maximal

Définition 2.1 Etant donné un graphe $G = (V, E)$, $M \subseteq E$ est un couplage si et seulement si $\delta_M(v) \leq 1$ pour tout noeud v.
Un chemin P est dit M-alternant s'il alterne entre une arête de M et une arêtre de $E \setminus M$.
Un chemin M-alternant P est dit M-augmentant s'il commence et termine par un noeud non couvert par M.

Théorème 2.1 Un couplage M est maximal si et seulement si il n'y a pas de chemin M-augmentant.

Démonstration. $\Leftarrow$ S'il y a un chemin M-augmentant

$\Rightarrow$ Supposons qu'il existe un M^* avec $|M^*| > |M|$ est M n'a pas de chemin augmentant. $M^* \otimes M$ est un sous-graphe de G avec degré dans $\{0, 1, 2\}$. On a plusieurs possiblités pour ce graphe (M^* est en rouge, M en bleu).



Toutefois, le quatrième type de graphe n'est pas possible puisque s'il y a plus de rouges que de bleus, on a un chemin augmentant pour M. Alors en suivant le

■

Théorème 2.2 On peut trouver un couplage maximal en $\mathcal{O}(mn)$.

2.1 Théorème de Kőnig

Théorème 2.3 — de Kőnig Dans un graphe biparti $G = (A \sqcup B, E)$:

$$\max_{couplage} |M| = \min_{\text{Vertex Cover}} |C|$$

Avant de prouver le théorème, une observation :

Proposition 2.1 — Dualité Faible. Dans un graphe G (non nécessairement biparti), on a :

$$|M| \leq |C|$$

Démonstration. Puisque les arêtes de M ne couvrent qu'au plus une fois chaque sommet, en particulier :

$$\sum_{v \in C} 1 \geq \sum_{(u,v) \in M} (1+1) \geq \sum_{e \in M} 1$$

■

Démonstration du Théorème de Kőnig. Soit M un couplage maximal de G. On pose U l'ensemble des sommets non atteints par M dans A et Z l'ensemble des sommets connecté par un chemin M alternant aux sommets de U. On pose $S = Z \cap X$ et $T = Z \cap Y$. Alors, chaque sommet de T est atteint par M et T est l'ensemble des voisins de S. Posons $K = (X \setminus S) \cup T$. Chaque arête de G a une de ses extrémités dans K. Donc K est une couverture des sommets de G et $|M| = |K|$ d'où le résultat. ■

2.2 Dualité et Programmation Linéaire

Définition 2.2 La programmation linéaire s'intéresse à la maximisiation de ${}^t wX$ en vérifiant $X \leq 0$ et $AX \leq b$ (problème de packing) ou à la minimisation de $b^T Y$ en vérifiant ${}^t Ay \geq w$ avec $y \geq 0$ (problème de covering).

Quelques exemples :

■ **Exemple 2.1 — Couplage Maximal comme Programmation Linéaire.** On cherche à maximiser $\sum_e X_e$ en ayant $\forall v \in A \cup B \sum_{e \in \delta(v)} X_e \leq 1$ et $\forall e \in E, X_e \geq 0$. On a donc ici A la matrice $(A_{v \in V, e \in E})$ d'incidence du graphe et b et w le vecteur Attila. On essaye de mettre autant d'arêtes que possible (packing).

Ce problème peut donc aussi s'écrire comme la minimisation de $\sum_{v \in A \cup B} Y_v$ en ayant $Y_u + Y_v \geq 1 \forall e = (u,v) \in E$, $Y_u \geq 0 \forall u \in A \cup B$. On essaye de mettre aussi peu d'arêtes que possible (covering). ■

Proposition 2.2 La dualité faible de la programmation linéaire est : $\forall x, y$ qui vérifie les contraintes, on a :

$${}^t wX \leq {}^t bY$$

Démonstration.

$${}^t wx \leq \left({}^t YA\right) X = {}^t Y \left(AX\right) \leq {}^t Yb = {}^t bY$$

■

Ceci redémontre la dualité faible pour les graphes.

Théorème 2.4 — Dualité Forte Il existe X^*, Y^* tels que ${}^t wX^* = {}^t bY^*$.

La dualité permet de faire les liens entre Max-Flow et Min-Cut ainsi qu'entre Max-Matching et Min-Vertex-Cover.

2.3 Couplage Pondéré dans un graphe biparti

Définition 2.3 Soit un graphe $G = (A \cup B, E = A \times B)$ avec $|A| = |B|$ et une fonction de poids entière sur les arêtes, on dit que $\pi : A \cup B \to \mathbb{N}$ est une couverture si $\forall e \in E, \pi(a) + \pi(b) \geq w(e)$.

Théorème 2.5 — Egrevary On a :

$$\max_{M \text{ couplage parfait}} w(M) = \min_{\pi \text{ couverture entière}} \sum_{v \in A \cup B} \pi(v)$$

Proposition 2.3 — Dualité Faible. On a : $w(M) \leq \sum \pi(v)$.

Démonstration. De même que précédemment en remplaçant par l'inégalité d'une couverture. ■

Démonstration du Théorème d'Egevary. Soit π une couverture minimale. Posons $G_\pi \subseteq G$ où une arête $e = (a, b)$ est dans G_π si et seulement si $\pi(a) + \pi(b) = w(e)$. C'est le sous-graphe des arêtes qu'on ne peut pas modifier. Soit M le couplage maximal dans G_π. Si $|M| = |A| = |B|$ alors on a fini puisqu'on a toujours égalité dans la preuve de la dualité faible. Sinon, si M n'est pas parfait, on définit π' comme suit :

$$\begin{cases} \forall a \in A_1 & \pi'(a) = \pi(a) - 1 \\ \forall b \in B_1 & \pi'(b) = \pi(b) + 1 \\ \forall v \notin A_1 \cup B_1 & \pi'(v) = \pi(v) \end{cases}$$

où A_1 est l'ensemble des sommets atteignables depuis un sommet non atteint pas M passant par un chemin augmentant et B_1 est l'ensemble des sommets couplés à des sommets de A_1. On définit également B_2 l'ensemble des sommets de B atteignables depuis un sommet non atteint par M en passant par un chemin M-augmentant, A_2 l'ensemble des sommets couplés à des sommets de B_2 et A_3, B_3 le reste de A et B. Puisque les arêtes qui ne sont pas dans G_ε vérifient $\pi(a) + \pi(b) \geq w((a, b)) + 1$, on a toujours $\pi'(a) + \pi'(b) \geq w(e)$. De plus, on a bien $\sum \pi' < \sum \pi$ et π' couvre toutes les arêtes, mais π' peut être négative. Si $\pi'(a_0) = -1$, alors on définit π'' comme :

$$\begin{cases} \pi''(a) = \pi'(a) + 1 & \forall a \in A \\ \pi''(b) = \pi'(b) - 1 & \forall b \in B \end{cases}$$

Ceci garantit que tous les $\pi''(a)$ sont non négatifs et que $\sum \pi'' = \sum \pi'$. Par ailleurs les $\pi''(b)$ sont aussi non négatifs. En effet, si on arrive à celà, on a une arête d'un poids $0 \leq w(e) \leq -1 + 0$. Donc π'' est une couverture plus petite que π. ■

Ceci nous donne par ailleurs un algorithme pour calculer une couverture minimale/un couplage maximal. Toutefois, celui-ci effectue au plus $|A| \max_e w(e) = \frac{nw}{2}$ itérations et est donc pseudo polynomial. Par ailleurs, cet algorithme ne termine pas nécessairement sur les réels.

La version ci-dessous, appelée méthode hongroise, termine quant à elle sur les réels et est plus efficace :

Théorème 2.6 Cette méthode termine et est demande un temps $\mathcal{O}(n^2 nn)$.

Démonstration.

- La boucle prend un temps $\mathcal{O}(m)$ pour s'effectuer.

Algorithme 5 Méthode Hongroise (Kuhn 1957)

$M \leftarrow \varnothing$, $\pi(a) \leftarrow \max_{e \in \delta(a)} w(e)$ et $\pi(b) \leftarrow 0$
while M n'est pas parfait **do**
  Construire G_π.
  Trouver le couplage maximal M dans G_π (par augmentation) et définir $A_1, A_2, A_3, B_1, B_2, B_3$
  Poser $\Delta \leftarrow \min_{a \in A_1, b \in B_2 \cup B_3} \pi(a) + \pi(b) - w((a,b))$
  $\pi(a) \leftarrow \pi(a) - \Delta$ et $\pi(b) \leftarrow \pi(b) + \Delta$
  $\delta \leftarrow \min_{a \in A} \pi(a)$.
  if $\delta < 0$ **then**
    $\pi(a) \leftarrow \pi(a) - \delta$ et $\pi(b) \leftarrow \pi(b) + \delta$

- On peut l'effectuer au plus n fois (lorsqu'$|A_1|$ augmente) et n fois lorsque $|M|$ augmente. Puisque ces deux effets peuvent se combiner, on a bien le résultat.

■

2.4 Relaxation et Optimalité

Proposition 2.4 — Relaxation. Étant donné un problème de programmation linéaire dual, X^* et Y^* sont tous les deux optimaux si et seulement si :

$$X_i^* \Longrightarrow \left({}^tAY^*\right)_i = W_i$$
$$Y_j^* \Longrightarrow \left(AX^*\right)_j = b_j$$

Démonstration. On a :

$${}^tWX \leq {}^tYAX \leq {}^tYb$$

Il y a égalité si et seulement si les contraintes sont serrées. D'où le résultat. ■

On peut appliquer ceci aux problèmes de couplage maximal : Si une arête est utilisée ($X_e > 0$), alors $Y_u + Y_v = 1$ et donc elle est la seule à atteindre l'une de ses extrémités.

On peut appliquer de même cela à Max-Flow Min-Cut

2.5 Forêts Hongroises et Floraisons

Définition 2.4 — Forêt Hongroise. Dans un graphe biparti, on construit une forêt dite hongroise en partant des sommets non couplésde B, en considérant ses voisins. Si ses voisins ne sont pas dans le couplage, on a un chemin augmentant. Sinon on ajoute les deux arêtes à la forêt et le nouveau sommet de B à la liste des sommets à considérer et on itère en évitant les sommets déjà atteints (puisque ça ne sert à rien). On appelle l'étape d'agrandissement de la forêt le *probing*.

On regarde toutes les arêtes au plus une fois, ce qui donne une complexité $\mathcal{O}(|E|)$.

Proposition 2.5 Si on ne trouve pas de chemin augmentant, on a un couplage maximal.

On va s'intéresser aux floraisons (cycles de longueur impaire) qui peuvent apparaître en créant l'équivalent d'une forêt hongroise dans un graphe non biparti. Ceci nous mène à l'algorithme suivant.

Algorithme 6 Edmonds Blossom Algorithm

Initialization (étant donné un couplage initial M)
$A = \varnothing$
$B = \{v \in V \mid \nexists u, (u, v) \in M\}$ (ce seront les racines de la forêt hongroise)

Boucle Tant qu'il existe $u \in B$ tel que (u, v) n'a pas été probée :

1. Ou bien $v \in A$ et on ne fait rien
2. Ou bien $v \notin A \cup B$, auquel cas v est couplé à w et on ajoute (u, v) et (v, w) à la forêt, en posant $A = A \cup \{v\}$ et $B = B \cup \{w\}$.
3. Ou bien $v \in B$ et :
   (a) ou bien u et v sont dans des arbres différents : on augmente (en utilisant le chemin de la racine de u à la racine de v) et on recommence l'algorithme en rouvrant toutes les floraisons
   (b) ou bien on contracte la floraison constituées des chemins de u et v à leur plus petit ancêtre commun (LCA) et (u, v). On traite ensuite cette floraison comme un sommet dans B.

Théorème 2.7 L'algorithme d'Edmonds trouve le couplage maximal.

Proposition 2.6 Le graphe après contraction a un chemin augmentant si et seulement si le graphe pré contraction a un chemin augmentant.

Démonstration. On note G' le graphe formé après la contraction d'une floraison dans G.

- Considérons un chemin augmentant dans G' qui passe par la contraction. On a deux cas :
  1. Si la floraison est une des extrémités du chemin, on sait que dans la floraison (qui a $2k + 1$ sommets) il n'y aura qu'un noeud d'attache. Ce sommet n'est pas dans M car il s'agit d'une arête finale d'un chemin augmentant dans G'. Ainsi, si le sommet est la base de la floraison, le chemin s'arrête et on renvoit le chemin original. Sinon, on peut étendre notre chemin selon des arêtes alternants jusqu'à avoir atteint le sommet de base.
  2. Sinon, on définit v_L et v_R les sommets adjacents à la floraison. On peut toujours prendre un chemin de v_L à v_R de longueur paire. Ainsi, on trouve un chemin augmentant dans G, puisque seulement l'une des deux v_L et v_R est couplée à la floraison et donc à la base de la floraison.
- Si on a chemin augmentant dans G qui passe par la floraison, on enlève un nombre pair d'arêtes du chemin donc le chemin dans G' reste un chemin augmentant.

■

On définit quelques invariants pour l'analyse de l'algorithme :

- Tous les sommets de A sont reliés à des sommets de B
- Une floraison est toujours dans B
- Si G' est le graphe dérivé d'une suite de contraction de floraisons, alors G' a un chemin augmentant si et seulement si G a un chemin augmentant. Ceci découle trivialement de 2.6.
- Si on trouve un chemin augmentant dans G' alors on peut rouvrir toutes les floraisons pour augmenter la taille de M dans G.
- Chaque noeud dans B (resp. A) peut revenir à sa racine par un chemin alternant de longueur paire (resp. impaire).

Soit U l'ensemble des sommets qui ne sont pas dans la forêt hongroise.

Démonstration de 2.7. **Preuve 1** On montre qu'à la fin de cet algorithme, dans G' (possiblement après beaucoup de contractions), il n'y a pas de chemin augmentant.

À la fin de l'algorithme d'Edmonds, il n'y a pas d'arêtes dans $B \times (B \cup U)$ (il peut y avoir des arêtes dans $A \times (A \cup U)$). Sinon, on aurait un moyen d'ajouter une arête à la forêt hongroise. Il n'y a donc pas de chemin augmentant dans G'.

Preuve 2 Par le théorème de Tutte-Berge.

Théorème 2.8 — Tutte-Berge Minimax

$$\max_{M \ couplage} |M| = \min_{A \subseteq V} \frac{|V| - (oc(V \setminus A) - |A|)}{2}$$

où $oc(V \setminus A)$ est le nombre de composantes connexes de taille impaire de $V \setminus A$

L'algorithme d'Edmonds trouve un certain A tel que :

$$|U| = oc(V \setminus A) - |A|$$

En effet, tout élément de A est matché à un élément de B non racine. Les éléments non matchés étant les racines d'arbres de la forêt, ils sont dans B. En enlevant les éléments de A, on crée une composante connexe de taille impaire par élément de B. Comme par ailleurs, $|B| - |A|$ est exactement l'ensemble des sommets non couplés, on a le résultat. En a donc bien le résultat.

■

Théorème 2.9 Cet algorithme demande un temps en $\mathcal{O}(mn^2)$.

Démonstration. La construction d'une forêt hongroise se fait en $\mathcal{O}(m)$.
La construction d'une floraison se fait en au plus $\mathcal{O}(m)$, et peut se faire au plus $\mathcal{O}(n)$. Ceci signifie qu'en temps au plus $\mathcal{O}(nm)$ il y a une augmentation. Puisqu'on ne peut pas augmenter plus de n fois, on a bien le résultat. ■

On peut améliorer la complexité de cet algorithme. Batinski a donné un algorithme pour une complexité en $\mathcal{O}(n^3)$ et Tarjan a donné un algorithme en $\mathcal{O}(nm \log n)$. Il existe un algorithme en $\mathcal{O}(\sqrt{n}m)$ datant de 1980. On a désormais des algorithmes en $\mathcal{O}(\sqrt{n}m \log_n \frac{n^2}{m})$ et $\mathcal{O}(n^w)$ la complexité optimale de multiplication de matrices.

2.6 Algorithmes de Streaming

Définition 2.5 Un algorithme de streaming est un algorithme qui ne peut pas stocker en mémoire toute son entrée en même temps.

On se limite ici à un algorithme qui ne peut stocker que $\mathcal{O}(n\text{polylog}(n))$ arêtes pour trouver un couplage maximal.

Théorème 2.10 Il y a un algorithme de streaming qui nécessite $\mathcal{O}\left(\frac{n}{\varepsilon} \log W\right)$ d'espace et renvoie une $\left(\frac{1}{2} - \varepsilon\right)$-approximation.

Algorithme 7 Algorithme de Streaming pour les Couplages Maximaux

Démonstration. **Initialisation** On pose $\pi(u) = 0$ pour tout $u \in V$ On crée une pile S vide.

Phase de Streaming Tant qu'une nouvelle arête $e = (a, b)$ arrive : Si $w(e) > (1 + \varepsilon)(\pi(a) + \pi(b))$:

- $\Delta = w(e) - (1 + \varepsilon)(\pi(a) + \pi(b))$
- $\pi(a) += \Delta$
- $\pi(b) += \Delta$

Phase Finale On effectue un algorithme glouton inverse en enlevant les arêtes de S une par une et en les ajoutant dans M si possible.

Lemme 2.1 On a :

$$\left(\sum_{v\in V}\pi(v)\right)(1+\varepsilon)\geq \max_{M\ couplage} w(M)$$

Démonstration. On a le problème de programmation linéaire suivant :

$$\begin{aligned}\max \sum_{e\in E} w(e)\chi_e\\ \forall v\in V \sum_{e\in\delta(v)}\chi_e\geq 1\\ \forall e\in E\chi_e\geq 0\end{aligned}$$

dont le dual est :

$$\begin{aligned}\min\sum_{v\in V}\pi(v)\\ \pi(a)+\pi(b)\geq w(e)\forall e=(a,b)\in E\\ \pi(v)\geq 0\forall v\in V\end{aligned}$$

On a le résultat par dualité faible en vérifiant que $(1+\varepsilon)\pi$ convient. ■

Lemme 2.2 Le couplage résultat M vérifie l'inégalité :

$$w(M)\geq\frac{1}{2}\left(\sum_{v\in V}\pi(v)\right)$$

Démonstration. L'algorithme va créer un pile de poids pour chaque noeuds (les Δ consécutifs des arêtes mettant en jeu le sommet). En particulier ■

On a alors :

$$\begin{aligned}w(M)&\leq\frac{1}{2(1+\varepsilon)}opt\\ &\geq\left(\frac{1}{2}-3\varepsilon\right)opt\end{aligned}$$

Pour ce qui est l'espace, il y a toujours au plus $\log_{1+\varepsilon} W\ \Delta$ dans la fonction de poids, et on a n sommets. En effet, à chaque fois qu'on ajoute un Δ, la valeur augmente de $1+\varepsilon$ au moins. On a le résultat en prenant un développement limité. ■

3 Global Min-Cut

On s'intéresse au problème suivant :

Global Min-Cut	
Input:	$G=(V,E)$ avec une fonction de capacité $E\to\mathbb{R}^+$
Output:	$\varnothing\subsetneq U\subsetneq V$ tel que $c(\delta(U))$ est minimal.

Dans le cas d'un graphe orienté, on cherche à minimiser $c(\delta^+(U))$.

Il est clair que résoudre le problème dans le cadre d'un graphe orienté suffit. On va simplifier le problème pour résoudre le problème suivant :

Minimal s-Cut	
Input:	$G = (V, E)$ avec une fonction de capacité $E \to \mathbb{R}^+$, $s \in V$
Output:	$\varnothing \subsetneq U \subsetneq V$ tel que $s \in U$ et $c(\delta(U))$ est minimal.

Proposition 3.1 On peut résoudre Global Min-Cut en résolvant deux fois Minimal s-cut, pour tout s.

Démonstration. On résout le problème pour un certain s, puis on le résout sur le graphe renversé. ■

Similairement, on va trouver un autre problème plus simple encore pour résoudre Minimal s-cut.

Minimal X-t-cut	
Input:	$G = (V, E)$ avec une fonction de capacité $E \to \mathbb{R}^+$, $t \in V$ et $X \subsetneq V \setminus \{t\}$
Output:	$\varnothing \subsetneq U \subseteq V$ tel que $X \subseteq U \subseteq V \setminus \{t\}$ et $c(\delta(U))$ est minimal.

Proposition 3.2 Si on résout Minimal X-t-cut n fois, on peut résoudre Minimal s-cut.

On propose pour cela l'algorithme suivant :

Algorithme 8 Minimal s-cut par Minimal X-t-cut

Initialisation Soit $X = \{s\}$.

Boucle Tant que $X \neq V$, on choisit $t \in V \setminus X$ arbitrairement. Soit $S_{|X|}$ le résultat de Minimal X-t-cut. On augmente $X = X \cup \{t\}$

Renvoi On renvoie $\operatorname{argmin}_{1 \leq t \leq |X|-1} S_t$.

Démonstration. Considérons la première itération (notée γ) où un sommet de l'autre côté de la s-cut optimale est choisi pour t. Alors $c(\delta^+(S_\gamma) \leq c(\delta^+(\text{opts} - \text{cut})$ et $c(\delta^+(\text{opts} - \text{cut}) \leq c(\delta^+(S_\gamma))$ car les deux coupes vérifient les deux propriétés. ■

On va pouvoir résoudre toutes les Minimal X-t-cut en résolvant un seul problème de flot. On rappelle que $exces(v) = \sum_{e \in \delta^-(v)} f(e) - \sum_{e \in \delta^+(v)} f(e)$. On va construire un algorithme se basant sur l'algorithme Preflow Push 2 qui va vérifier les deux invariants suivants :

1. Sur $V \setminus X$, $exces(v) \geq 0$.
2. Pour tout $v \in X$, $d(v) = n$, $\forall u, v \in G(f), d(u) \leq d(v) + 1$, $d(t) \leq |X| - 1$ et $d(t) \leq d(u)$.

Définition 3.1 Le seau associé à k est l'ensemble $B(k) = \{v \mid d(v) = k\}$. k est un niveau de coupe si pour tout $u \in B(k), (u, v) \in G(f), d(v) \geq d(u)$.

Proposition 3.3 Si k est un niveau de coupe, l'ensemble des u tels que $d(u) \geq k$ est une coupe dans le graphe résiduel $G(f)$.

Proposition 3.4 Soit $k < n$ un niveau de coupe. Si pour tout $v \neq t$ tel que $d(v) < k$ on a $\text{exces}(v) = 0$, alors $\{u \mid d(u) \geq k\}$.

Démonstration. Débrouille-toi, normalien ! ■

Lemme 3.1 Les seaux non vides (de niveau $< n$) sont toujours consécutifs.

Algorithme 9 Hao-Ortin — X-preflow-push

Initialisation On pose $X = \{s\}$, $d(s) = n$, $d(v) = 0, \forall v \neq t$ où $t \in V \setminus \{s\}$, $l = n-1$, $f(e) = c(e), \forall e \in \delta^+(s)$.

Boucle Tant que $X \neq V$, on applique l'algorithme preflow-push (ou push-relabel) avec 3 modifications :

1. Un sommet $u \neq t$ est actif si et seulement si $d(u) < l$ et exces$(u) > 0$.
2. Si u va être ré-étiqueté et que $|B(d(u))| = 1$, on ne ré-étiquette pas u et on pose $l = d(u)$.
3. Si u est ré-étiqueté pour avoir $d(u) > l$, on pose $l = n-1$.

On pose $S_{|X|} = \{u \mid d(u) \geq l\}$, $X = X \cup \{t\}$, $d(t) = n$ et $\forall e \in \delta^+(t), f(e) = c(e)$. On choisit pour le nouveau t celui celui avec le minimum de $d(t)$ parmi les sommets de $V \setminus X$. Si $d(t) \geq l$ on pose en plus $l = n-1$.

Renvoi On renvoie $\operatorname{argmin}_{1 \leq \alpha \leq |X|-1} c(\delta^+ S_\alpha)$.

Démonstration. Il est clair que le lemme est vrai initialement. Par induction, il y a trois dangers possibles :

1. Un sommet est retiré par ré-étiquetage du niveau k. Ceci ne peut pas arriver puisqu'on ne ré-étiquette pas le dernier sommet d'un seau.
2. Un sommet est ajouté par ré-étiquetage au niveau k. Par définition, on ne peut pas aller plus haut que le minimum plus un.
3. Un t est ajouté à X. Puisque l'hypothèse d'induction est valide, et puisqu'on a toujours $d(v) >= d(t)$, le seau en $d(t)-1$ est vide et : ou bien tous les seaux autres sont vides, ou bien le seau en $d(t)+1$ est non vide et l'hypothèse reste vraie.

■

Lemme 3.2 On a toujours $d(t) \leq |X| - 1$.

Démonstration. Soit t' le nouveau t choisi. On a $d(t') \leq d(t) + 1$ par le lemme 3.1. Par induction, $d(t') \leq d(t) - 1 \leq |X| = |X \cup \{t\}| - 1$ ■

Le lemme précédent montre que l'algorithme 9 vérifie les invariants précédents.

Lemme 3.3 Pour $u \notin X$, on a $d(u) \leq n - 2$

Démonstration. Soit v le sommet de $V \setminus X$ avec la plus grande étiquette. Par le lemme 3.1, il y a au plus $n - |X| - 1$ seaux entre $d(t)$ et $d(v$ d'où, par le lemme 3.2, $d(u) \leq d(t) + n - |X| - 1 \leq n - 1 - 1 = n - 2$. ■

Corollaire 3.1 $n-1$ est un niveau de coupe.

Lemme 3.4 On a toujours $d(t) < l \leq n - 1$ et l est toujours un niveau de coupe.

Démonstration. On va procéder par induction :

1. Dans le cas on prend $l = n - 1$, par 3.1, on a le résultat.
2. Dans le cas où on pose $l = d(u)$, puisque u est le dernier de son seau, tous ses voisins vérifient $d(v) \geq d(u) > d(t)$ d'où le résultat.

■

Théorème 3.1 La cut de renvoi est une coupe s-minimale.

Démonstration. Lorsque l'algorithme s'arrête, tous les sommets d'étiquette $< l$ sont inactifs et ont donc exces $= 0$, d'où le résultat par la proposition 3.4 ■

Théorème 3.2 L'algorithme demande un temps $\mathcal{O}(n^2 m)$.

Démonstration.
- Les *vrais* ré-étiquetages (ceux où $d(u)$ croît) demandent un temps $\mathcal{O}(n^2)$
- Les *faux* ré-étiquetages (ceux où on change la valeur de l) demandent un temps $\mathcal{O}(n^2)$: en effet chaque fois qu'on ré-étiquette un sommet autre que t, son étiquette augmente d'au moins 1 et l'augmentation est inférieure à $n-2$. L'étiquette de t est n à la fin de l'itération et n'augmente pas plus.
- Les pushs saturants et non-saturants ne voient pas leur complexité changer.

■

On peut trouver un algorithme un peu plus efficace en $\mathcal{O}(n^2 m \log_n \frac{n^2}{m})$.

3.1 Karger

On présente ici un algorithme qui renvoie une cut correcte avec une bonne probabilité.

Algorithme 10 Karger

Tant que $|V| > 2$, on choisit $e = (u, v)$ uniformément au hasard. On contracte e en un sommet et on renvoie la cut définie par les 2 sommets restants.

Lemme 3.5 Étant donné une coupe optimale $\left(S, \bar{S}\right)$, la probabilité de prendre une arête dans $E \cap \left(S \times \bar{S}\right)$ est au plus

Démonstration. On a : $opt \leq \min_{v \in V} d(v) = \delta$. Ainsi, $\mathbb{P}(\text{échec}) \leq \frac{\delta}{|E|} \leq \frac{\delta}{\delta |V|/2} \leq \frac{2}{|V|}$. ■

Lemme 3.6 La probabilité qu'on ne prenne jamais d'arête dans $E \cap \left(S \times \bar{S}\right)$ est $\geq \frac{1}{\binom{n}{2}} \geq \frac{2}{n^2}$.

Démonstration. Par le lemme 3.5, la probabilité p vérifie :

$$\begin{aligned} p \geq & \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right) \cdots \left(1 - \frac{2}{3}\right) \\ = & \frac{n-2}{n}\frac{n-3}{n-1}\frac{n-4}{n-2} \cdots \frac{2}{4}\frac{1}{3} \\ = & \frac{2}{n(n-1)} = \frac{1}{\binom{n}{2}} \end{aligned}$$

■

Théorème 3.3 Si on répète l'algorithme de Karger $10 \times n^2 \log n$ fois, on réussit au moins une fois avec probabilité $1 - \frac{1}{n^{\mathcal{O}(1)}}$

Démonstration. La probabilité p d'échec vérifie :

$$p \leq \left(1 - \frac{2}{n^2}\right)^{10 n^2 \log(n)} \leq \left(e^{-1}\right)^{5 \log n} = \frac{1}{n^5}$$

■

Théorème 3.4 Le nombre de coupes minimales est inférieur à $\binom{n}{2}$ et cette borne est optimale.

Démonstration. Toutes les preuves précédentes étant vraies pour une coupe minimale quelconque, en sommant les probabilités on a le résultat. En prenant un cycle on a le résultat, puisqu'une coupe est équivalente à choisir deux sommets ■

4 Matroïdes

On notera ici $I + x = I \cup \{x\}$ et $I - x = I \setminus \{x\}$.

Définition 4.1 $M = (U, \mathbb{I}), \mathbb{I} \subseteq 2^U$ est un système indépendant si :

A_0 $\varnothing \in \mathbb{I}$

A_1 $u \subseteq V \in \mathbb{I} \Rightarrow u \in \mathbb{I}$.

Les éléments de $\mathbb{I}$ sont appelés ensembles indépendants. Si de plus, M vérifie l'une des conditions équivalentes suivantes, M est appelé un matroïde :

A_2 Pour $\bar{U} \subseteq U$ tous les ensembles indépendants maximaux dans U ont la même taille.

A_2' Étant donné $I, J \in \mathbb{I}$, si $|I| > |J|$ il existe $x \in I \setminus J$ tel que $J + x \in \mathbb{I}$.

A_2' Pour $I, J \in I$ tels que $|J \setminus I| = 2$ et $|I \setminus J| = 1$ alors il existe $x \in J \setminus I$ tel que $I + x \in \mathbb{I}$.

Les matroïdes ont été introduits dans différents domaines de manière indépendante dans les années 30.

■ **Exemple 4.1 — Matroïde de Graphe.** Étant donné un graphe (V, E), en considérant $U = E$ et $\mathbb{I}$ l'ensemble des forêts, $(U, \mathbb{I})$ est un matroïde. En effet, toutes les forêts maximales sont de taille $|U| - \text{cc}(G)$ où $\text{cc}(G)$ est le nombre de composantes connexes de G. ■

■ **Exemple 4.2 — Matroïde Linéaire.** Étant donné une matrice $A = (C_1, \ldots, C_n)$, U l'ensemble de ses colonnes et $\mathbb{I}$ l'ensemble des sous familles des colonnes linéairement indépendantes forment un matroïde. En effet, cela à revient à dire que toutes les bases d'un même espace ont même cardinal. ■

■ **Exemple 4.3 — Matroïde de Partitions.** Étant donné un ensemble $U = \sqcup U_i$, on prend $\mathbb{I}$ l'ensemble des ensembles dont l'intersection avec chacun des U_i est de cardinal au plus 1. ■

Définition 4.2

- Le rang de $V \subseteq U$, noté $\gamma_M(V)$ l'entier $\max_{I \in \mathbb{I} \cap P(V)} |I|$.
- Si $|I| = \gamma_M(V)$, I est une base de V.
- Si $|I| = \gamma_M(U)$, I est une base.
- $C \subseteq U$ est un circuit si C est un ensemble dépendant minimal.
- On appelle l'ensemble engendré par V l'ensemble $\text{span}_M(V) = \{e \mid \gamma_M(V) = \gamma_M(V + e)\}$.

Entrée:	Matroïde $M = (U, \mathbb{I})$ avec une fonction de poids $w : U \to \mathbb{R}$
Sortie:	Ensemble indépendant de poids maximal.

Algorithme 11 Algorithme Glouton – Edmonds

$I \leftarrow \varnothing$
while $\exists x \in U \setminus I$ tel que $w(x) > 0$ et $I + x \in \mathbb{I}$ **do**
  $I = I + e$ où $e = \text{argmax}\, w(e)$

Théorème 4.1 — Rado - Edmonds L'algorithme glouton renvoie une solution optimale.

Démonstration. Supposons que l'algorithme renvoie $I = \{e_1, \cdots, e_h\}$ (les éléments étant ordonnés par ordre décroissants de poids). Supposons qu'on ait un ensemble $OPT = \{f_1, \cdots, f_s\}$ de poids optimal. Supposons $w(OPT) > w(I)$. Alors, on choisit le plus petit l tel que $w(f_l) > w(e_l)$. S'il n'en existe pas, prenons $l = h$. Alors, $\{e_1, \ldots, e_{l-1}\}$ est un ensemble indépendent maximal dans $\{e_1, \ldots, e_{l-1}, f_1, \ldots, f_l\}$. Or, $f_1, \ldots, f_l$ est indépendant : on a une contradiction par la propriété A_2 des matroïdes. ■

La preuve de cette algorithme nous donne une nouvelle condition équivalente pour la définition d'un matroïde :

Proposition 4.1 L'algorithme glouton renvoie un ensemble indépendant de poids maximal pour toute fonction de poids si et seulement si $(U, \mathbb{I})$ est un matroïde.

Démonstration. Il nous reste à montrer que cette propriété implique l'une des propriétés précédentes. Supposons que la propriété ci-dessus soit vérifiée mais que $(U, \mathbb{I})$ ne soit pas un matroïde. Il existe I, J maximaux tels que $|I| > |J|$. Alors, on pose $w(e) = 1$ pour $e \in I \cup J$, $w(e) = 0$ sinon. L'algorithme glouton ne spécifiant pas comment résoudre les conflits d'égalité, on pourrait avoir J en résultat. ■

Intersection de Matroïdes	
Entrée:	$M_1 = (U, \mathbb{I}_1), M_2 = (U, \mathbb{I}_2)$
Sortie:	Le plus grand $I \in \mathbb{I}_1 \cap \mathbb{I}_2$

On peut voir le problème de couplage dans les graphes bipartis ou le problème de couverture par un arbre coloré comme des problèmes d'intersection de matroïdes.

Dans le cas du problème de couverture d'un arbre coloré par exemple, on prend le matroïde du graphe et le matroïde de partition sur les arêtes en considérant la partition par couleurs. Pour la couverture par un arbre dirigé, on prend le matroïde du graphe et le matroïde de la partition des arêtes selon leur destination.

On note que l'intersection de deux matroïdes n'est pas un matroïde. On note par ailleurs que résoudre l'intersection de 3 matroïdes permet de résoudre des problèmes NP-Complets (réduction aux Chemins Hamiltonniens, ou 3D-Matching)

Lemme 4.1 Si $I \in \mathbb{I}$, $I + x$ crée au plus un circuit C.

Démonstration. Par l'absurde, soit A le plus petit contre-example. $A+x$ contient deux circuits C_1, C_2. On peut supposer $A + x = C_1 \cup C_2$. Il existe par ailleurs $a \in C_1 \setminus C_2$ et $b \in C_2 \setminus C_1$. Considérons $A' = A + x - a - b$. Alors $A' \in I$ et A et A' sont tous deux des ensembles indépendants maximaux dans $A + x$. ■

Corollaire 4.1 Si $I \in \mathbb{I}$, $I + x$ contient C, alors $I + x - y \in \mathbb{I}$, pour tout $y \in C$.

Définition 4.3 On définit le graphe auxiliaire $G(I) = (I \sqcup U \setminus I, E)$ d'un ensemble indépendant $I \in \mathbb{I}_1 \cap \mathbb{I}_2$ où E est un ensemble d'arêtes orientées défini par :

- Si $I + x$ contient un circuit C_2 dans M_2, on met une arête de y vers x pour tout $y \in C_2 - x$.
- Si $I + x$ contient un circuit C_1 dans M_1, on met une arête de x vers y pour tout $y \in C_1 - x$.

On définit de plus $X_2 = \{e \mid I + e \in \mathbb{I}_2\}$ et $X_1 = \{e \mid I + e \in \mathbb{I}_1\}$ des parties de $U \setminus I$.

Algorithme 12 Edmonds

$I \leftarrow \varnothing$
while il existe un chemin de X_2 vers X_1 dans $G(I)$ **do**
  Trouver le plus court chemin p
  $I = I \oplus P$

Théorème 4.2 Quand l'algorithme s'arrête, I est le plus grand ensemble indépendent commun.

Lemme 4.2 Soit $P = e_0 f_1 e_1 f_2 \cdots f_t e_t$ le plus court chemin dans $G(I)$ de X_2 vers X_1. Soit $I_0 = I + e_0$, $I_t = I_0 + \{e_0, \ldots, e_t\} - \{f_1, \ldots, f_t\}$ Alors pour tout i, $I_i \in \mathbb{I}$.

Démonstration. Par induction sur i, $I_0 \in \mathbb{I}_2$ par définition de X_2. Pour $i > 0$, $I_{j-1} + f_j$ pour $j \geq i$ a le même circuit que $I + f_j$ car aucun des circuits dans $I + e_z$ pour $z < j$ n'implique d'éléments de $\{f_1, \ldots, f_{z-1}\}$ puisque P est le plus court

chemin. En effet, sinon, on aurait un raccourci et une arête d'un certain f_k vers e_z, ce qui permettrait de raccourcir P. ■

Lemme 4.3 On a $I \oplus P \in \mathbb{I}_1$.

Démonstration. En inversant les indices dans la preuve précédente. ■

Théorème 4.3 — Minimax d'Edmonds On a :

$$\max_{I \in \mathbb{I}_1 \cap \mathbb{I}_2} |I| = \min_{A \subseteq U} \gamma_1(A) + \gamma_2(U \setminus A)$$

Démonstration. Prouvons d'abord la dualité faible : Par définition du rang*, $I \cap A$ étant indépendant dans M_1 et $I \cap U \setminus A$ étant indépendant dans M_2, $|I \cap A| \leq \gamma_1(A)$ et $|I \cap U \setminus A| \leq \gamma_2(U \setminus A)$. La dualité forte découlera de la correction de l'algorithme d'Edmonds. ■

Démonstration de la Correction de l'Algorithme d'Edmonds. Quand l'algorithme s'arrête, il n'y a pas de chemin. Soit R l'ensemble des sommets atteignables depuis X_2. Alors $R \cap I$ est une base de R dans M_1.

Démonstration. En effet, $R \setminus I \cap X_1 = \varnothing$ donc, pour $e \in R \setminus I$, $I + e$ a un circuit C. Or, C n'implique que e et $R \cap I$, puisque $G(I)$ est biparti. Ceci prouve bien que $R \cap I$ est de cardinal maximal et donc une base. ■

Par symétrie, $I \setminus R$ est une base de $R^{\complement}$ dans M_2. Ainsi, $|I| = |R \cap I| + |I \setminus R| = \gamma_1(R) + \gamma_2\left(R^{\complement}\right)$. Donc I fait égalité dans la formule d'Edmonds. ■

Théorème 4.4 L'algorithme demande un temps polynomial.

Définition 4.4 Étant donné un matroïde $M = (U, \mathbb{I})$, on peut définir un autre matroïde appelé *matroïde dual* $M^* = (U, \mathbb{I}^*)$ où $\mathbb{I}^* = \{A \mid U \setminus A \text{ contient une base dans } M\}$.

Pour le matroïde graphique, c'est l'ensemble des ensembles d'arêtes qu'on peut retirer en gardant chaque composante du graphe connexe. Un circuit devient donc une coupe.

Démonstration. Étant donné $A \subseteq U$, on pose I l'ensemble indépendent maximal dans A considéré dans $M^* = (U, \mathbb{I}^*)$. On choisit une base X de $U \setminus A$ dans M. Notons $B = X + A \setminus I$. Alors, B est une base de U dans M. En effet : CÉPAFINI ■

5 Arbres de Gomory-Hu

Définition 5.1 Étant donné un graphe non-orienté $G = (V, E)$, pondéré par $u : E \to \mathbb{R}^+$, soit $\lambda(u, v)$ la valeur d'une coupe minimum de u, v dans G. On définit aussi $\lambda = \min_{(u,v) \in V} \lambda(u, v)$. On appelle $\lambda(u, v)$ la connectivité de u, v.

On s'intéresse au problème de calculer toutes les connectivités. Il est clair qu'on peut les calculer en $\binom{2}{|V|}$ calculs de max-flow-min-cut. En réalité, $|V| - 1$ suffisent. Pour celà, on va construire un arbre $T = (V, F)$ sur le graphe $G = (V, E)$, avec une fonction $l : F \to \mathbb{R}^+$ de sorte qu'on conserve toutes les informations de coupe dans l'arbre au sens où la valeur de $\lambda(a, b)$ sera l'étiquette minimale d'une arête du chemin de a à b dans F, et que si on retire cette arête, on obtient les deux ensembles de sommets de la coupe.

Définition 5.2 — Arbres de Gomory-Hu. Soit $G = (V, E)$, $w : E \to \mathbb{R}^+$ notre graphe d'entrée. $T = (V, F), l : F \to \mathbb{R}^+$ est un arbre de Gomory-Hu si, étant donnés $a, b \in V$, on pose e l'arête de plus faible $l(e)$ sur le chemin de a à b dans T vérifie $\lambda(a, b) = l(e)$. De plus, en notant $s(e)$ une des composantes de $T - e$, alors $\lambda(a, b) = u(\delta_G(s(e)))$.

Lemme 5.1 $T = (V, F)$ est un arbre de Gomory-Hu si et seulement si :

$$\forall (i, j) \in F, \lambda(i, j) = l(i, j) = u(\delta_G(s(i, j)))$$

Démonstration. Soit $P = v_1, \ldots, v_k$ le chemin de a à b dans T. On va montrer que $\min_{i \in [\![1,k-1]\!]} \lambda(v_i, v_{i+1}) \leq \lambda(a, b) \leq \min_{i \in [\![1,k-1]\!]} \lambda(v_i, v_{i+1})$. En effet, toute coupe sur ce chemin sépare a et b, et donc montrer cette séquence d'inégalités montre le lemme. Il suffit de montrer la première inégalité, la seconde étant triviale par l'argument précédent. Soit S^* la coupe $a - b$ minimale dans G. Supposons que $a \in S^*$. S^* doit séparer le chemin puisqu'elle sépare a, b. En particulier, si v_{i+1} est le premier sommet du chemin qui n'est pas dans S^*, alors S^* est une coupe pour $v_i - v_{i+1}$. Donc en particulier, $\lambda(i, j) \leq u(\delta(S^*))$. ■

Définition 5.3 — Fonctions Sous-Modulaires. Une fonction $f : 2^V \to \mathbb{R}^+$ est dite *sous-modulaire* si $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$ pour toutes parties A, B de V.

Lemme 5.2 La fonction de coupe $u(\delta(\cdot))$ est sous-modulaire.

Démonstration. On regarde tous les cas possibles pour une arête dont au moins l'une des extrémités est dans A ou dans B. ■

Lemme 5.3 Si f est symétrique et sous-modulaire, i.e. $f(S) = f(V \setminus S)$, alors $f(A) + f(B) \geq f(A \setminus B) + f(B \setminus A)$.

Démonstration. Débrouille-toi, normalien. ■

Lemme 5.4 Si $a, b, c \in V$, $\lambda(a, c) \geq \min\{\lambda(a, b), \lambda(b, c)\}$

Démonstration. On a soit : $\lambda(a, c) \geq \lambda(a, b)$ soit $\lambda(a, c) \geq \lambda(b, c)$. ■

Corollaire 5.1 Le minimum de $\{\lambda(a, b), \lambda(b, c), \lambda(a, c)\}$.

Démonstration. En faisant tourner les valeurs dans le lemme précédent. ■

Lemme 5.5 Soit R la coupe $r - s$ minimale, pour $r \in R$, et soit X la coupe $s - t$ minimum pour $s \in X$. On suppose que $t \notin R$.

1. Si $r \in X$, alors $R \cap X$ est une coupe $r - s$ minimale et $R \cup X$ est une coupe $s - t$ minimale.
2. Si $r \notin X$, alors $R \setminus X$ est une coupe $r - s$ minimale et $X \setminus R$ est une coupe $s - t$ minimale. De plus $\lambda(r, s) = \lambda(r, t)$.

Démonstration.

1. $u(\delta(R)) + u(\delta(X)) \geq u(\delta(R \cup X)) + u(\delta(R \cup X)) + u(\delta(R \cap X))$. La première inégalité vient de la sous-modularité, et la deuxième inégalité découpe du fait que les deux termes du deuxième membre soient des coupes.
2. $u(\delta(R)) + u(\delta(X)) \geq u(\delta(R \setminus X)) + u(\delta(X \setminus R)) \geq u(\delta(R)) + u(\delta(X))$.

■

Lemme 5.6 À tout moment, $\forall (r_i, r_j) \in T, \lambda(i, j) = l(r_i, r_j) = u(\delta(s(r_i, r_j)))$.

Démonstration. Par induction, la seule chose qu'il suffit de remarquer c'est que si on sépare r en r, t, on ne touche pas aux autres sommets. Sinon, supposons $(r_i, r_j) \in T$ avant la séparation de r_i.

Algorithme 13 Construction de l'Arbre Gomory-Hu

$\mathbb{V} \leftarrow \{V\}$, soit z un représentant arbitraire de V, $T \leftarrow z$.
while $\exists V_i \in \mathbb{V}, |V_i| \geq 2$ **do**
  Soit r_i le représentant de V.
  Soit $t \in V, t \neq r_i$
  Trouver la coupe $r_i - t$ minimum X.
  Diviser V_i en $V_i \cap X, V_i \setminus X$.
  $V_i \cap X$ a r_i pour représentant, $V_i \setminus X$ a t pour représentant.
  Séparer le noeud r_i de T en deux noeuds r_i et t, connectés par (r_i, t) avec $l(r_i, t) = \lambda(r_i, t)$.
  Pour tous $(r_i, r_j) \in T$, si $(r_j \notin X)$, remplacer (r_i, r_j) dans T par (r_j, t)

1. Si $r_j \notin X$ alors r_j, t est une nouvelle arête dans T. Par le cas 2 de 5.5 en posant $r_i = s$, $t = t$ et $r_j = r$, alors $\lambda(r_i, r_j) = \lambda(r_j, t)$ et la nouvelle arête définit une coupe $r_j - t$ minimale. De plus, $X \setminus R$ est toujours une coupe $r_j - t$ minimale.
2. Si $r_j \in X$, alors (r_j, r_i) est toujours une arête de T. En posant à nouveau $r = r_j, t = t, s = r_i$, par le cas 1 de 5.5, $X \cup R$ est une coupe $r_j - t$ minimale.

■

Théorème 5.1 L'algorithme précédent permet de construire l'arbre de Gomory-Hu d'un graphe en $|V| - 1$ recherches de coupe minimale.

■ **Exemple 5.1**

- Étant donné un multi-graphe $G = (V, E)$ dont tous les sommets sont de degré au moins k, il existe $a, b \in V$ tels que a et b sont connectés par au moins k chemins disjoints. Ceci se prouve en prenant l'arbre de Gomory-Hu et en trouvant une arête de poids au moins k.
- Étant donné un graphe $G = (V, E)$, $c : E \to \mathbb{R}^+$, choisir $F \subseteq E$ de poids minimal de tel sorte que $G \setminus F$ a au moins k composantes. On donne un algorithme de 2-approximation pour ce problème dans 14

■

Algorithme 14 2-Approx pour k-Cut

Construire T l'arbre de Gomory-Hu pour G.
Choisir les $k - 1$ arêtes de T d'étiquettes minimales.
On coupe le graphe original G en se basant sur ces $k - 1$ arêtes.

5.1 k-Couverture

Étant donné un univers $U = \{e_1, \ldots, e_m\}$ et une famille $F \subseteq 2^U$, on veut choisir k ensembles $S_1, \ldots, S_k$ dans F tels que $\left|\bigcup_{i=1}^{k} S_i\right|$ est maximale. Ce problème est un cas particulier du problème de maximisation sous contrainte d'une fonction sous-modulaire. On peut en effet définir une fonction de couverture $g : 2^F \to N$ par les éléments couverts par les sommets $S \subseteq F$, cette fonction étant sous-modulaire.

Algorithme 15 Algorithme Glouton

$X_0 \leftarrow \varnothing$
$i \leftarrow 1$
while $i \leq k$ **do**
  $X_i = X_0 + \arg\max |A| \cap U \setminus \cup_{S \in X_i} S$

Théorème 5.2 Cet algorithme renvoie une $1 - \frac{1}{e}$-approximation.

Démonstration. On a :

$$g(X_1) - g(X_0) \geq \frac{OPT}{k} = \left(1 - \left(1 - \frac{1}{k}\right)\right) OPT$$

puis :

$$g(X_2) - g(X_1) \geq \frac{OPT - g(X_1)}{k}$$

et donc :

$$g(X_2) \geq OPT(1 - \left(1 - \frac{1}{k}\right)^2)$$

Par induction :

$$g(X_i i) \geq OPT(1 - \left(1 - \frac{1}{k}\right)^i)$$

D'où :

$$g(X_k) \geq OPT(1 - \left(1 - \frac{1}{k}\right)^k) \geq OPT(1 - \frac{1}{e})$$

■

Théorème 5.3 — **Feige** S'il y avait un algorithme polynômial qui garantissait $\left(1 - \frac{1}{e} + \varepsilon\right)$-approximation, alors $P = NP$.

■ **Exemple 5.2 — Quelques Cas de Fonctions Sous-Modulaires.**

- Les fonctions linéaires
- La fonction rang d'un matroïde : c'est la formule de Grassmann.
- La fonction d'addition de budget : $f(S) = \min\{B, \sum_{i \in S} a_i\}$.
- Étant donné deux matroïdes M_1, M_2, la fonctions $A \mapsto \gamma_1(A) + \gamma_2(S \setminus A)$

Pour ces fonctions sous-modulaires, le problème de minimization peut être résolu en temps polynomial. ■