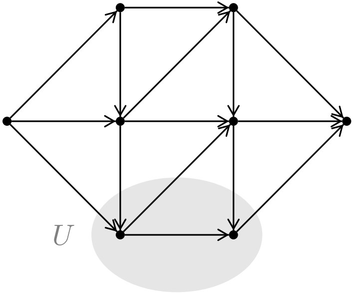
Arcs sortants et entrants

Définition

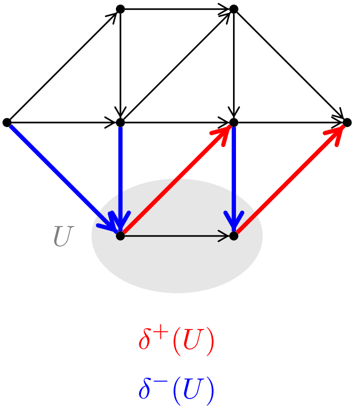
Soient $G = (V, E)$ un graphe orienté et $U \subseteq V$. Alors, on note $\partial^+(U)$ l'ensemble des arcs dont le début est dans U. De même, on note $\partial^-(U)$ l'ensemble des arcs dont la fin est dans U.



Arcs sortants et entrants

Définition

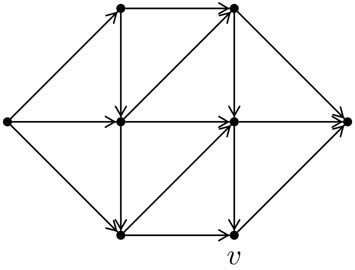
Soient $G = (V, E)$ un graphe orienté et $U \subseteq V$. Alors, on note $\partial^+(U)$ l'ensemble des arcs dont le début est dans U. De même, on note $\partial^-(U)$ l'ensemble des arcs dont la fin est dans U.



Degré sortant et degré entrant

Définition

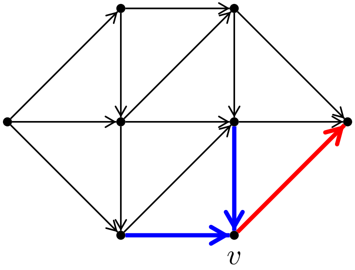
- le *degré entrant* $d^-(v)$, est le nombre d'arcs dont la fin est v.
- le *degré sortant* $d^+(v)$, est le nombre d'arcs dont le début est v.
- On a $d^-(v) = |\partial^-(\{v\})|$ et $d^+(v) = |\partial^+(\{v\})|$.
- v est une *source* si $d^-(v) = 0$.
- v est un *puits* si $d^+(v) = 0$.



Degré sortant et degré entrant

Définition

- le *degré entrant* $d^-(v)$, est le nombre d'arcs dont la fin est v.
- le *degré sortant* $d^+(v)$, est le nombre d'arcs dont le début est v.
- On a $d^-(v) = |\partial^-(\{v\})|$ et $d^+(v) = |\partial^+(\{v\})|$.
- v est une *source* si $d^-(v) = 0$.
- v est un *puits* si $d^+(v) = 0$.



$d^+(v) = 1$

$d^-(v) = 2$

Flot max et coupe min

- Deux problèmes classiques de l'optimisation combinatoire.
- Exemple de la dualité mathématique.

Quelques applications

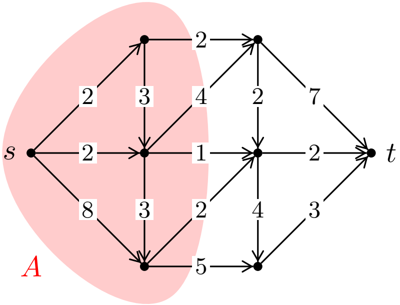
- Fouille de données.
- Sélection de projets.
- Ordonnancement (par exemple compagnies aériennes).
- Segmentation de l'image.
- Connectivité et fiabilité des réseaux.
- Calcul distribuée. . .

Réseaux

Définition

Un *réseau* est un graphe orienté $G = (V, E)$ avec :

- deux sommets spéciaux :
  - la source s tel que $d^-(s) = 0$
  - le puits t tel que $d^+(s) = 0$
- une fonction de *capacité* $c : E \to \mathbb{R}^+$.



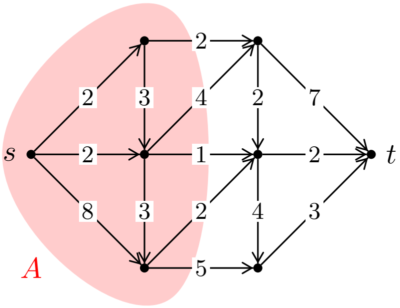
Coupes

Définition

Une $s-t$ *coupe* est une partition (A, B) de V telle que $s \in A$ et $t \in B$

Définition

La *capacité* d'une coupe (A, B) est $\text{cap}(A, B) = \sum_{e \in \delta^+(A)} c(e)$.

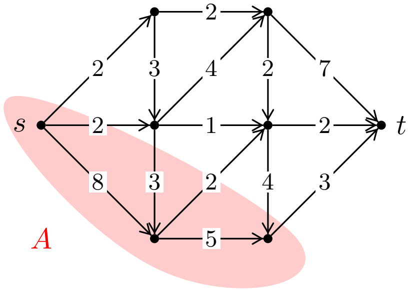


$$\text{cap}(A, B) = 2 + 4 + 1 + 2 + 5 = 14$$

Le problème de la coupe minimum

Problème

Trouver une $s-t$ coupe de capacité minimum.



$$\text{cap}(A, B) = 2 + 2 + 2 + 3 = 9$$

Flots

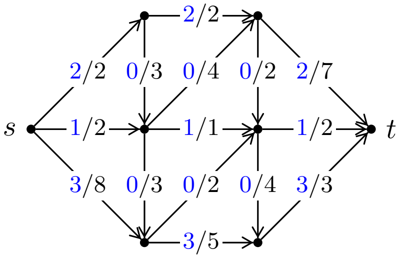
Définition

Un $s - t$ *flot* est une fonction $f : E \to \mathbb{R}^+$ qui vérifie

- Pour tout $e \in E$, $0 \leq f(e) \leq c(e)$ (*contrainte de capacité*)
- Pour tout $v \in V - \{s, t\}$, $\sum_{e \in \delta^-(v)} f(e) = \sum_{e \in \delta^+(v)} f(e)$ (*conservation de flot*).

Définition

La *valeur* d'un flot f est $\mathrm{val}(f) = \sum_{e \in \delta^+(s)} f(e)$.

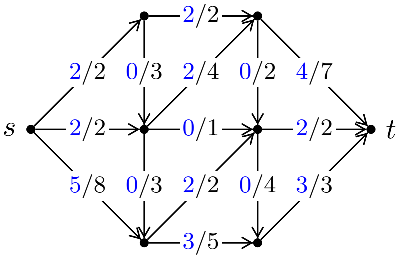


$\mathrm{val}(f) = 2 + 1 + 3 = 6$

Le problème du flot maximum

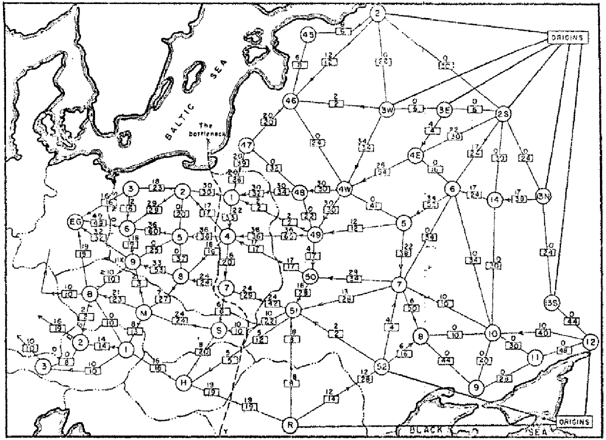
Problème

Trouver un $s - t$ flot de valeur maximum.



$$\mathrm{val}(f) = 2 + 2 + 5 = 9$$

Un peu d'histoire : le réseau ferroviaire soviétique

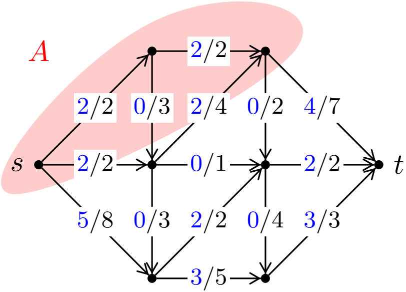


Flots et coupes

Lemme

Soit f un flot et (A, B) une $s-t$ coupe. Alors, le flot qui traverse la coupe est égal au flot qui sort de s.

$$\sum_{e\in\delta^+(A)} f(e) - \sum_{e\in\delta^-(A)} f(e) = \mathrm{val}(f).$$



$$\begin{aligned}\mathrm{val}(f) &= 2+2+5\\ &= 5+2-2+4\end{aligned}$$

Flots et coupes

Démonstration

$$
\begin{aligned}
\mathrm{val}(f) &= \sum_{e\in\delta^+(s)} f(e) \\
&= \sum_{e\in\delta^+(s)} f(e) - \sum_{e\in\delta^-(s)} f(e) \\
&= \sum_{v\in A}\left(\sum_{e\in\delta^+(v)} f(e) - \sum_{e\in\delta^-(v)} f(e)\right) \\
&= \sum_{e\in\delta^+(A)} f(e) - \sum_{e\in\delta^-(A)} f(e).
\end{aligned}
$$

Dualité faible

Lemme

Soit f un flot et (A, B) une $s - t$ coupe quelconques. Alors, la valeur de f est inférieure ou égale à la capacité de la coupe.

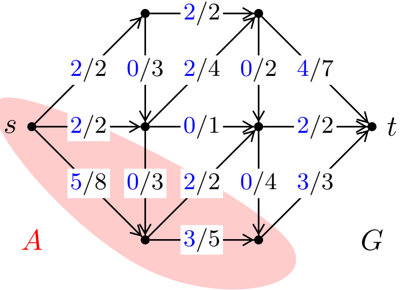
Démonstration

$$\begin{aligned}\mathrm{val}(f) &= \sum_{e \in \delta^+(A)} f(e) - \sum_{e \in \delta^-(A)} f(e) \\ &\leq \sum_{e \in \delta^+(A)} f(e) \\ &\leq \sum_{e \in \delta^+(A)} c(e) \\ &= \mathrm{cap}(A, B).\end{aligned}$$

Certificat d'optimalité

Corollaire

Soit f un flot et (A, B) une coupe. Si $\text{val}(f) = \text{cap}(A, B)$, alors f est un flot max et (A, B) est une coupe min.



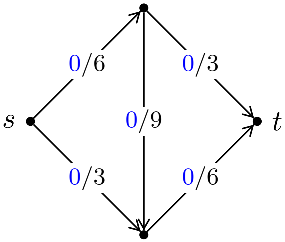
$$\text{val}(f) = 9$$

$$\text{cap}(A, B) = 9$$

Vers un algorithme de flot max

Un algorithme glouton

- Commencer par le flot nul, càd, $f(e) = 0$ pour chaque arc $e \in E$.
- Trouver un $s - t$ chemin P où tout arc vérifie $f(e) < c(e)$.
- Augmenter le flot le long le chemin P.
- Répéter jusqu'à devenir coincé.

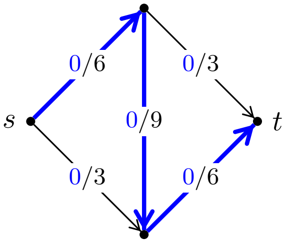


$\mathrm{val}(f) = 6$

Vers un algorithme de flot max

Un algorithme glouton

- Commencer par le flot nul, càd, $f(e) = 0$ pour chaque arc $e \in E$.
- Trouver un $s - t$ chemin P où tout arc vérifie $f(e) < c(e)$.
- Augmenter le flot le long le chemin P.
- Répéter jusqu'à devenir coincé.

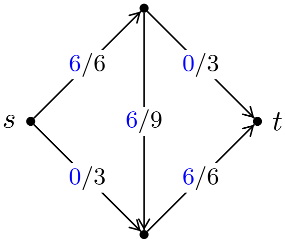


$\mathrm{val}(f) = 6$

Vers un algorithme de flot max

Un algorithme glouton

- Commencer par le flot nul, càd, $f(e) = 0$ pour chaque arc $e \in E$.
- Trouver un $s - t$ chemin P où tout arc vérifie $f(e) < c(e)$.
- Augmenter le flot le long le chemin P.
- Répéter jusqu'à devenir coincé.



$\mathrm{val}(f) = 6$

Le graphe résiduel

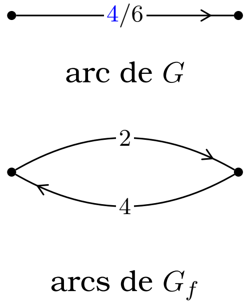
Arc originel :

- $e = (u, v) \in E$.
- Flot $f(e)$, capacité $c(e)$.

Arc résiduel :

- "annuler" le flot envoyé.
- $e = (u, v)$ et $e^R = (v, u)$.
- Capacité résiduelle :

$$c_f(e) = \begin{cases} c(e) - f(e) & \text{si } e \in E \\ f(e) & \text{si } e^R \in E. \end{cases}$$



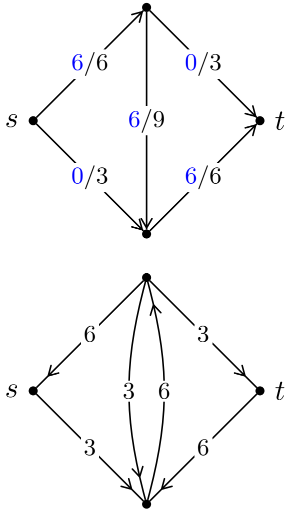
Le graphe résiduel

Graphe résiduel :

- Arcs résiduels avec capacité résiduelle positive.
- $E_f = \{e \mid f(e) < c(e)\} \cup \{e^R \mid f(e) > 0\}$.

Chemin augmentant :

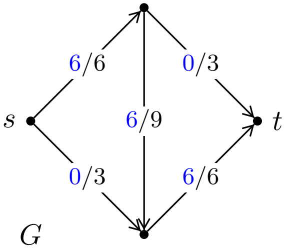
- Un *chemin augmentant* est un s–t chemin simple dans le graphe résiduel G_f.
- La *capacité* $\mathrm{cap}(G_f, P)$ d'un chemin augmentant P est le minimum des capacités résiduelles parmi tous les arcs de P.

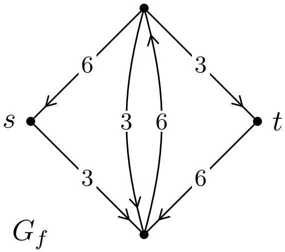


Propriété clé

- Soit f un flot et P un chemin augmentant dans G_f.
- En envoyant un flot de valeur $\mathrm{cap}(G_f, P)$ on obtient un nouveau flot f' de valeur $\mathrm{val}(f') = \mathrm{val}(f) + \mathrm{cap}(G_f, P)$.

```
Augment(f,c,P):
    ε ← min{c_f(e) : e ∈ E(P)}
    for e ∈ E(P):
        if e ∈ E:
            f(e) ← f(e) + ε
        else:
            f(e^R) ← f(e) − ε
    return f
```

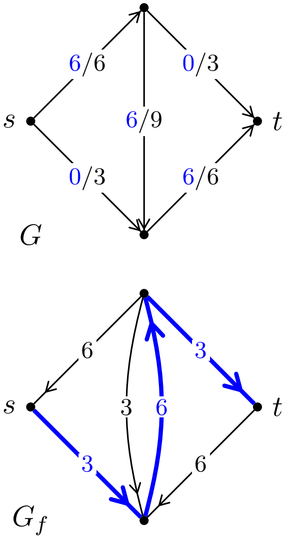




Propriété clé

- Soit f un flot et P un chemin augmentant dans G_f.
- En envoyant un flot de valeur $\text{cap}(G_f, P)$ on obtient un nouveau flot f' de valeur $\text{val}(f') = \text{val}(f) + \text{cap}(G_f, P)$.

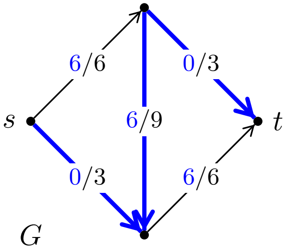
```
Augment(f,c,P):
    ε ← min{c_f(e) : e ∈ E(P)}
    for e ∈ E(P):
        if e ∈ E:
            f(e) ← f(e) + ε
        else:
            f(e^R) ← f(e) − ε
    return f
```

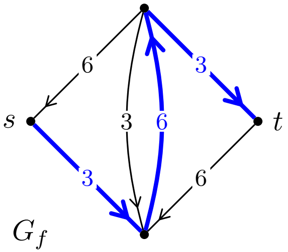


Propriété clé

- Soit f un flot et P un chemin augmentant dans G_f.
- En envoyant un flot de valeur $\text{cap}(G_f, P)$ on obtient un nouveau flot f' de valeur $\text{val}(f') = \text{val}(f) + \text{cap}(G_f, P)$.

```
Augment(f,c,P):
    ε ← min{c_f(e) : e ∈ E(P)}
    for e ∈ E(P):
        if e ∈ E:
            f(e) ← f(e) + ε
        else:
            f(e^R) ← f(e) − ε
    return f
```

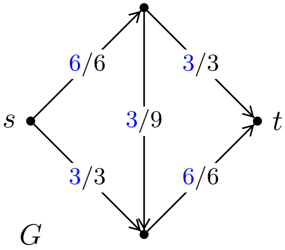


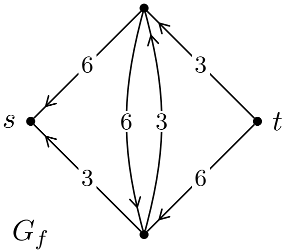


Propriété clé

- Soit f un flot et P un chemin augmentant dans G_f.
- En envoyant un flot de valeur $\mathrm{cap}(G_f, P)$ on obtient un nouveau flot f' de valeur $\mathrm{val}(f') = \mathrm{val}(f) + \mathrm{cap}(G_f, P)$.

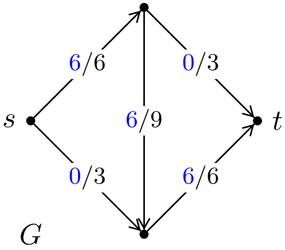
```
Augment(f,c,P):
    ε ← min{c_f(e) : e ∈ E(P)}
    for e ∈ E(P):
        if e ∈ E:
            f(e) ← f(e) + ε
        else:
            f(e^R) ← f(e) − ε
    return f
```

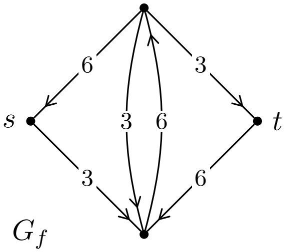




Algorithme de Ford–Fulkerson

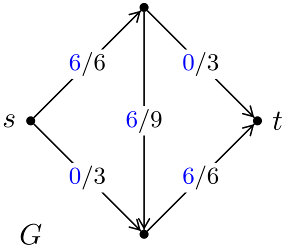
```
Ford-Fulkerson(G, s, t, c):
    for e ∈ E(G):
        f ← 0
    G_f ← graphe residuel
    while ∃ chemin augmentant P:
        f ← Augment(f, c, P)
        mettre a jour G_f
    return f
```

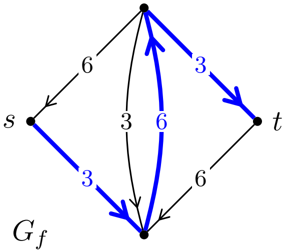




Algorithme de Ford–Fulkerson

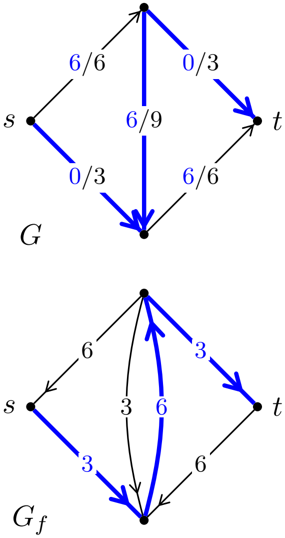
```
Ford-Fulkerson(G, s, t, c):
    for e ∈ E(G):
        f ← 0
    G_f ← graphe residuel
    while ∃ chemin augmentant P:
        f ← Augment(f, c, P)
        mettre a jour G_f
    return f
```





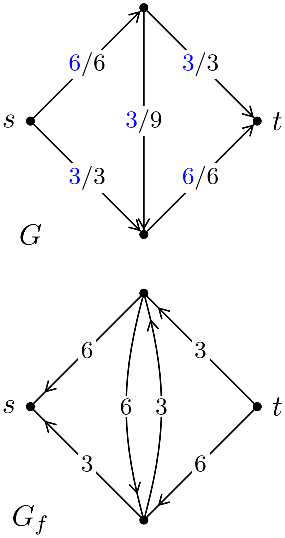
Algorithme de Ford–Fulkerson

```
Ford-Fulkerson(G, s, t, c):
    for e ∈ E(G):
        f ← 0
    G_f ← graphe residuel
    while ∃ chemin augmentant P:
        f ← Augment(f, c, P)
        mettre a jour G_f
    return f
```



Algorithme de Ford–Fulkerson

```
Ford-Fulkerson(G, s, t, c):
    for e ∈ E(G):
        f ← 0
    G_f ← graphe residuel
    while ∃ chemin augmentant P:
        f ← Augment(f, c, P)
        mettre a jour G_f
    return f
```



Le théorème flot-max/coupe-min

Théorème des chemins augmentants

Un flot f est maximum ssi il n'y a pas de chemin augmentant.

Théorème (Elias–Feinstein–Shannon 1956 ; Ford–Fulkerson 1956)

La valeur maximum d'un flot est égale à la capacité minimum d'une coupe.

On va prouver les deux théorèmes en même temps en démontrant que les énoncés suivants sont équivalents :

(1) Il existe une coupe (A, B) telle que $\mathrm{val}(f) = \mathrm{cap}(A, B)$.

(2) Le flot f est maximum.

(3) Il n'existe pas de chemin augmentant par rapport à f.

Démonstration du théorème flot-max/coupe-min (1/2)

(1) $\Rightarrow$ (2) Corollaire à la dualité faible.

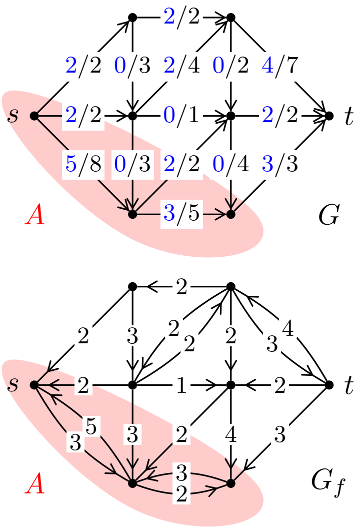
(2) $\Rightarrow$ (3) Soit f un flot. S'il existe un chemin augmentant P, on peut augmenter f en envoyant un flot le long P.

(3) $\Rightarrow$ (1)

- Soit f un flot sans chemin augmentant.
- Soit A un ensemble de sommets atteignables depuis s dans le graphe résiduel.
- Par la définition de A, $s \in A$.
- Par la définition de f, $t \notin A$.

Démonstration du théorème flot-max/coupe-min (2/2)

$$\begin{aligned}
\mathrm{val}(f) &= \sum_{e \in \delta^+(A)} f(e) - \sum_{e \in \delta^-(A)} f(e) \\
&= \sum_{e \in \delta^+(A)} c(e) \\
&= \mathrm{cap}(A, B)
\end{aligned}$$



Comment trouver une coupe minimum ?

- Il suffit de prendre A comme l'ensemble des sommets atteignables à partir de s dans le graphe résiduel G_f.
- C'est-à-dire, $v \in A$ ssi il existe un chemin orienté dans G_f avec sommet de départ s et sommet d'arrivée v.
- Si vous trouvez que $t \in A$, alors il existe un chemin augmentant et f n'est pas maximum — dans ce cas, il faut encore faire tourner Ford–Fulkerson !