Algorithmique des graphes

Flots et applications (2)

Anthony Labarre

31 mars 2021

Université Gustave Eiffel

Résumé des épisodes précédents

- On a défini la notion de flot dans un réseau, la notion de coupe, et le problème du flot maximum ;

Résumé des épisodes précédents

- On a défini la notion de flot dans un réseau, la notion de coupe, et le problème du flot maximum ;
- On a vu un théorème donnant l'équivalence entre trouver un flot maximum et trouver une coupe minimum ;

Résumé des épisodes précédents

- On a défini la notion de flot dans un réseau, la notion de coupe, et le problème du flot maximum ;
- On a vu un théorème donnant l'équivalence entre trouver un flot maximum et trouver une coupe minimum ;
- On a vu l'algorithme d'Edmonds-Karp, permettant de trouver un flot maximum en $O(|V||A|^2)$;

Résumé des épisodes précédents

- On a défini la notion de flot dans un réseau, la notion de coupe, et le problème du flot maximum ;
- On a vu un théorème donnant l'équivalence entre trouver un flot maximum et trouver une coupe minimum ;
- On a vu l'algorithme d'Edmonds-Karp, permettant de trouver un flot maximum en $O(|V||A|^2)$;
- Questions ?

Le menu du jour

- Un algorithme pour obtenir une coupe minimum ;

Le menu du jour

- Un algorithme pour obtenir une coupe minimum ;
- La preuve du théorème *max-flow min-cut*, justifiant la correction de la méthode de Ford-Fulkerson ;

Le menu du jour

- Un algorithme pour obtenir une coupe minimum ;
- La preuve du théorème *max-flow min-cut*, justifiant la correction de la méthode de Ford-Fulkerson ;
- L'algorithme de Dinitz, plus rapide que Edmonds-Karp ;

Le menu du jour

- Un algorithme pour obtenir une coupe minimum ;
- La preuve du théorème *max-flow min-cut*, justifiant la correction de la méthode de Ford-Fulkerson ;
- L'algorithme de Dinitz, plus rapide que Edmonds-Karp ;
- Le calcul d'un couplage maximum dans un graphe biparti ;

Obtention d'une coupe minimum

- Le théorème *max-flow min-cut* nous donne l'équivalence entre un flot maximum et une coupe minimum ;

Obtention d'une coupe minimum

- Le théorème *max-flow min-cut* nous donne l'équivalence entre un flot maximum et une coupe minimum ;
- On a déjà vu comment calculer un flot maximum ;

Obtention d'une coupe minimum

- Le théorème *max-flow min-cut* nous donne l'équivalence entre un flot maximum et une coupe minimum ;
- On a déjà vu comment calculer un flot maximum ;
- Comment déduire explicitement une coupe minimum (S, T) ?

Obtention d'une coupe minimum

- Le théorème *max-flow min-cut* nous donne l'équivalence entre un flot maximum et une coupe minimum ;
- On a déjà vu comment calculer un flot maximum ;
- Comment déduire explicitement une coupe minimum (S, T) ?
- Il nous suffit de parcourir le résiduel final G_f :

Obtention d'une coupe minimum

- Le théorème *max-flow min-cut* nous donne l'équivalence entre un flot maximum et une coupe minimum ;
- On a déjà vu comment calculer un flot maximum ;
- Comment déduire explicitement une coupe minimum (S, T) ?
- Il nous suffit de parcourir le résiduel final G_f :
  - $S = s$ et tous ses descendants dans G_f ;

Obtention d'une coupe minimum

- Le théorème *max-flow min-cut* nous donne l'équivalence entre un flot maximum et une coupe minimum ;
- On a déjà vu comment calculer un flot maximum ;
- Comment déduire explicitement une coupe minimum (S, T) ?
- Il nous suffit de parcourir le résiduel final G_f :
  - $S = s$ et tous ses descendants dans G_f ;
  - $T = \bar{S} = V \setminus S$;

Obtention d'une coupe minimum

- Le théorème *max-flow min-cut* nous donne l'équivalence entre un flot maximum et une coupe minimum ;
- On a déjà vu comment calculer un flot maximum ;
- Comment déduire explicitement une coupe minimum (S, T) ?
- Il nous suffit de parcourir le résiduel final G_f :
  - $S = s$ et tous ses descendants dans G_f ;
  - $T = \bar{S} = V \setminus S$;

Proposition 1

Soit G un réseau de source s, f un flot pour ce réseau, G_f le réseau résiduel associé, et S l'union de s et de ses descendants dans G_f. Si G_f ne contient pas de chemin augmentant, alors $(S, \bar{S})$ est une coupe minimum pour G.

Résultats auxiliaires

Pour prouver la Proposition 1, on a besoin des résultats suivants :

Lemme 2

Soit f un flot sur un réseau de flot G, et (S, T) une coupe sur G. Alors $|f| = f(S, T)$.

Résultats auxiliaires

Pour prouver la Proposition 1, on a besoin des résultats suivants :

Lemme 2

Soit f un flot sur un réseau de flot G, et (S, T) une coupe sur G. Alors $|f| = f(S, T)$.

Corollaire 3

Pour tout flot f sur un réseau de flot G et pour toute coupe (S, T) sur ce même réseau, on a $|f| \leq c(S, T)$.

Résultats auxiliaires

Pour prouver la Proposition 1, on a besoin des résultats suivants :

Lemme 2

Soit f un flot sur un réseau de flot G, et (S, T) une coupe sur G. Alors $|f| = f(S, T)$.

Corollaire 3

Pour tout flot f sur un réseau de flot G et pour toute coupe (S, T) sur ce même réseau, on a $|f| \leq c(S, T)$.

Démonstration.

□

Résultats auxiliaires

Pour prouver la Proposition 1, on a besoin des résultats suivants :

Lemme 2

Soit f un flot sur un réseau de flot G, et (S, T) une coupe sur G. Alors $|f| = f(S, T)$.

Corollaire 3

Pour tout flot f sur un réseau de flot G et pour toute coupe (S, T) sur ce même réseau, on a $|f| \leq c(S, T)$.

Démonstration.

$$|f| \underset{\substack{\downarrow \\ \text{Lemme 2}}}{=} f(S, T)$$

$\square$

Résultats auxiliaires

Pour prouver la Proposition 1, on a besoin des résultats suivants :

Lemme 2

Soit f un flot sur un réseau de flot G, et (S, T) une coupe sur G. Alors $|f| = f(S, T)$.

Corollaire 3

Pour tout flot f sur un réseau de flot G et pour toute coupe (S, T) sur ce même réseau, on a $|f| \leq c(S, T)$.

Démonstration.

$$|f| \underset{\substack{\downarrow \\ \text{Lemme 2}}}{=} f(S, T) \underset{\substack{\downarrow \\ \text{définition}}}{=} \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{u \in S} \sum_{v \in T} f(v, u)$$

□

Résultats auxiliaires

Pour prouver la Proposition 1, on a besoin des résultats suivants :

Lemme 2

Soit f un flot sur un réseau de flot G, et (S, T) une coupe sur G. Alors $|f| = f(S, T)$.

Corollaire 3

Pour tout flot f sur un réseau de flot G et pour toute coupe (S, T) sur ce même réseau, on a $|f| \leq c(S, T)$.

Démonstration.

$$|f| \underset{\substack{\downarrow \\ \text{Lemme 2}}}{=} f(S, T) \underset{\substack{\downarrow \\ \text{définition}}}{=} \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{u \in S} \sum_{v \in T} f(v, u) \leq \sum_{u \in S} \sum_{v \in T} f(u, v)$$

$\square$

Résultats auxiliaires

Pour prouver la Proposition 1, on a besoin des résultats suivants :

Lemme 2

Soit f un flot sur un réseau de flot G, et (S,T) une coupe sur G. Alors $|f| = f(S,T)$.

Corollaire 3

Pour tout flot f sur un réseau de flot G et pour toute coupe (S,T) sur ce même réseau, on a $|f| \leq c(S,T)$.

Démonstration.

$$|f| \underset{\substack{\downarrow\\ \text{Lemme 2}}}{=} f(S,T) \underset{\substack{\downarrow\\ \text{définition}}}{=} \sum_{u\in S}\sum_{v\in T} f(u,v) - \sum_{u\in S}\sum_{v\in T} f(v,u) \leq \sum_{u\in S}\sum_{v\in T} f(u,v)$$

$$\text{contrainte capacité} \leftarrow \leq \sum_{u\in S}\sum_{v\in T} c(u,v)$$

□

Résultats auxiliaires

Pour prouver la Proposition 1, on a besoin des résultats suivants :

Lemme 2

Soit f un flot sur un réseau de flot G, et (S, T) une coupe sur G. Alors $|f| = f(S, T)$.

Corollaire 3

Pour tout flot f sur un réseau de flot G et pour toute coupe (S, T) sur ce même réseau, on a $|f| \leq c(S, T)$.

Démonstration.

$$|f| \underset{\text{Lemme 2}}{=} f(S, T) \underset{\text{définition}}{=} \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{u \in S} \sum_{v \in T} f(v, u) \leq \sum_{u \in S} \sum_{v \in T} f(u, v)$$

$$\underset{\text{contrainte capacité}}{\leq} \sum_{u \in S} \sum_{v \in T} c(u, v) \underset{\text{définition}}{=} c(S, T)$$

□

Preuve de la Proposition 1

1. $(S, \bar{S})$ **est une coupe**, car $s \in S$ et $t \in \bar{S}$ puisque t n'est pas un descendant de s ;

Preuve de la Proposition 1

1. $(S, \bar{S})$ **est une coupe**, car $s \in S$ et $t \in \bar{S}$ puisque t n'est pas un descendant de s ;
2. $c(S, \bar{S})$ **est minimum**, car : $\nexists\ (u, v) \in A(G_f) : u \in S, v \in \bar{S}$; et $\forall$ arc $(v, u) \in A(G_f)$ de $\bar{S}$ vers S, on a :

Preuve de la Proposition 1

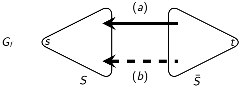
1. $(S, \bar{S})$ **est une coupe**, car $s \in S$ et $t \in \bar{S}$ puisque t n'est pas un descendant de s ;
2. $c(S, \bar{S})$ **est minimum**, car : $\nexists\ (u, v) \in A(G_f) : u \in S, v \in \bar{S}$; et $\forall$ arc $(v, u) \in A(G_f)$ de $\bar{S}$ vers S, on a :
   (a) soit $(v, u) \in A(G) \Rightarrow f(v, u) = 0$ **sinon** $(u, v) \in A(G_f)$;

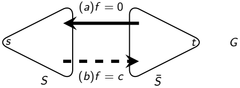
Preuve de la Proposition 1

1. $(S, \bar{S})$ **est une coupe**, car $s \in S$ et $t \in \bar{S}$ puisque t n'est pas un descendant de s ;
2. $c(S, \bar{S})$ **est minimum**, car : $\nexists\ (u, v) \in A(G_f) : u \in S, v \in \bar{S}$; et $\forall$ arc $(v, u) \in A(G_f)$ de $\bar{S}$ vers S, on a :
   - (a) soit $(v, u) \in A(G) \Rightarrow f(v, u) = 0$ **sinon** $(u, v) \in A(G_f)$;
   - (b) soit $(u, v) \in A(G) \Rightarrow f(u, v) = c(u, v)$ **sinon** $(u, v) \in A(G_f)$;

Preuve de la Proposition 1

1. $(S, \bar{S})$ **est une coupe**, car $s \in S$ et $t \in \bar{S}$ puisque t n'est pas un descendant de s ;
2. $c(S, \bar{S})$ **est minimum**, car : $\nexists\ (u, v) \in A(G_f) : u \in S, v \in \bar{S}$; et $\forall$ arc $(v, u) \in A(G_f)$ de $\bar{S}$ vers S, on a :
   - (a) soit $(v, u) \in A(G) \Rightarrow f(v, u) = 0$ **sinon** $(u, v) \in A(G_f)$;
   - (b) soit $(u, v) \in A(G) \Rightarrow f(u, v) = c(u, v)$ **sinon** $(u, v) \in A(G_f)$;





La valeur du flot est :

$$|f| = f(S, \bar{S})$$

$\downarrow$

Lemme 2

Preuve de la Proposition 1

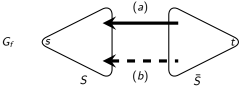
1. $(S, \bar{S})$ **est une coupe**, car $s \in S$ et $t \in \bar{S}$ puisque t n'est pas un descendant de s ;
2. $c(S, \bar{S})$ **est minimum**, car : $\nexists\ (u, v) \in A(G_f) : u \in S, v \in \bar{S}$; et $\forall$ arc $(v, u) \in A(G_f)$ de $\bar{S}$ vers S, on a :
   - (a) soit $(v, u) \in A(G) \Rightarrow f(v, u) = 0$ **sinon** $(u, v) \in A(G_f)$;
   - (b) soit $(u, v) \in A(G) \Rightarrow f(u, v) = c(u, v)$ **sinon** $(u, v) \in A(G_f)$;

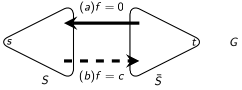
La valeur du flot est :

$$|f| \underset{\text{Lemme 2}}{=} f(S, \bar{S}) \underset{\text{définition}}{=} \sum_{u \in S} \sum_{v \in \bar{S}} f(u, v) - \sum_{u \in S} \sum_{v \in \bar{S}} f(v, u)$$

Preuve de la Proposition 1

1. $(S, \bar{S})$ **est une coupe**, car $s \in S$ et $t \in \bar{S}$ puisque t n'est pas un descendant de s ;
2. $c(S, \bar{S})$ **est minimum**, car : $\nexists\ (u, v) \in A(G_f) : u \in S, v \in \bar{S}$; et $\forall$ arc $(v, u) \in A(G_f)$ de $\bar{S}$ vers S, on a :
   - (a) soit $(v, u) \in A(G) \Rightarrow f(v, u) = 0$ **sinon** $(u, v) \in A(G_f)$;
   - (b) soit $(u, v) \in A(G) \Rightarrow f(u, v) = c(u, v)$ **sinon** $(u, v) \in A(G_f)$;





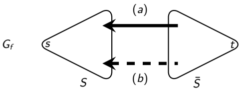
La valeur du flot est :

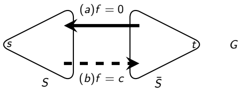
$$|f| = f(S, \bar{S}) = \sum_{u \in S} \sum_{v \in \bar{S}} f(u, v) - \sum_{u \in S} \sum_{v \in \bar{S}} f(v, u) = \sum_{u \in S} \sum_{v \in \bar{S}} f(u, v)$$

Lemme 2 définition arcs (a)

Preuve de la Proposition 1

1. $(S, \bar{S})$ **est une coupe**, car $s \in S$ et $t \in \bar{S}$ puisque t n'est pas un descendant de s ;
2. $c(S, \bar{S})$ **est minimum**, car : $\nexists\ (u, v) \in A(G_f) : u \in S, v \in \bar{S}$; et $\forall$ arc $(v, u) \in A(G_f)$ de $\bar{S}$ vers S, on a :
   - (a) soit $(v, u) \in A(G) \Rightarrow f(v, u) = 0$ **sinon** $(u, v) \in A(G_f)$;
   - (b) soit $(u, v) \in A(G) \Rightarrow f(u, v) = c(u, v)$ **sinon** $(u, v) \in A(G_f)$;





La valeur du flot est :

$$\underbrace{|f| = f(S, \bar{S})}_{\text{Lemme 2}} \underbrace{=}_{\text{définition}} \sum_{u \in S} \sum_{v \in \bar{S}} f(u, v) - \sum_{u \in S} \sum_{v \in \bar{S}} f(v, u) \underbrace{=}_{\text{arcs (a)}} \sum_{u \in S} \sum_{v \in \bar{S}} f(u, v) \underbrace{=}_{\text{arcs (b)}} \sum_{u \in S} \sum_{v \in \bar{S}} c(u, v)$$

Preuve de la Proposition 1

1. $(S, \bar{S})$ **est une coupe**, car $s \in S$ et $t \in \bar{S}$ puisque t n'est pas un descendant de s ;
2. $c(S, \bar{S})$ **est minimum**, car : $\nexists\ (u, v) \in A(G_f) : u \in S, v \in \bar{S}$; et $\forall$ arc $(v, u) \in A(G_f)$ de $\bar{S}$ vers S, on a :
   - (a) soit $(v, u) \in A(G) \Rightarrow f(v, u) = 0$ **sinon** $(u, v) \in A(G_f)$;
   - (b) soit $(u, v) \in A(G) \Rightarrow f(u, v) = c(u, v)$ **sinon** $(u, v) \in A(G_f)$;

La valeur du flot est :

$$|f| \underset{\text{Lemme 2}}{=} f(S, \bar{S}) \underset{\text{définition}}{=} \sum_{u \in S} \sum_{v \in \bar{S}} f(u, v) - \sum_{u \in S} \sum_{v \in \bar{S}} f(v, u) \underset{\text{arcs (a)}}{=} \sum_{u \in S} \sum_{v \in \bar{S}} f(u, v) \underset{\text{arcs (b)}}{=} \sum_{u \in S} \sum_{v \in \bar{S}} c(u, v)$$

...et $\sum_{u \in S} \sum_{v \in \bar{S}} c(u, v) = c(S, \bar{S})$. On a donc $|f| = c(S, \bar{S})$, et $(S, \bar{S})$ est donc minimum (voir Corollaire 3). $\square$

Preuve du théorème *max-flow min-cut*

Théorème 4 (Max-flow min-cut)

[1] Soit f un flot dans un réseau $G = (V, A, c)$ de source s et de puits t. Les conditions suivantes sont équivalentes :

Démonstration.

□

Les points 1 et 2 du théorème prouvent la correction de la méthode de Ford-Fulkerson.

Preuve du théorème *max-flow min-cut*

Théorème 4 (Max-flow min-cut)

[1] Soit f un flot dans un réseau $G = (V, A, c)$ de source s et de puits t. Les conditions suivantes sont équivalentes :

1. *f est un flot maximum ;*

Démonstration.

□

Les points 1 et 2 du théorème prouvent la correction de la méthode de Ford-Fulkerson.

Preuve du théorème *max-flow min-cut*

Théorème 4 (Max-flow min-cut)

[1] Soit f un flot dans un réseau $G = (V, A, c)$ de source s et de puits t. Les conditions suivantes sont équivalentes :

1. *f est un flot maximum ;*
2. *le réseau résiduel G_f ne contient pas de chemin augmentant ;*

Démonstration.

□

Les points 1 et 2 du théorème prouvent la correction de la méthode de Ford-Fulkerson.

Preuve du théorème *max-flow min-cut*

Théorème 4 (Max-flow min-cut)

[1] Soit f un flot dans un réseau $G = (V, A, c)$ de source s et de puits t. Les conditions suivantes sont équivalentes :

1. *f est un flot maximum ;*
2. *le réseau résiduel G_f ne contient pas de chemin augmentant ;*
3. *il existe une coupe (S, T) pour G telle que $|f| = c(S, T)$.*

Démonstration.

□

Les points 1 et 2 du théorème prouvent la correction de la méthode de Ford-Fulkerson.

Preuve du théorème *max-flow min-cut*

Théorème 4 (Max-flow min-cut)

[1] Soit f un flot dans un réseau $G = (V, A, c)$ de source s et de puits t. Les conditions suivantes sont équivalentes :

1. *f est un flot maximum ;*
2. *le réseau résiduel G_f ne contient pas de chemin augmentant ;*
3. *il existe une coupe (S, T) pour G telle que $|f| = c(S, T)$.*

Démonstration.

- 1. $\Rightarrow$ 2. : contraposée : un chemin augmentant permet d'augmenter f ;

□

Les points 1 et 2 du théorème prouvent la correction de la méthode de Ford-Fulkerson.

Preuve du théorème *max-flow min-cut*

Théorème 4 (Max-flow min-cut)

[1] Soit f un flot dans un réseau $G = (V, A, c)$ de source s et de puits t. Les conditions suivantes sont équivalentes :

1. *f est un flot maximum ;*
2. *le réseau résiduel G_f ne contient pas de chemin augmentant ;*
3. *il existe une coupe (S, T) pour G telle que $|f| = c(S, T)$.*

Démonstration.

- 1. $\Rightarrow$ 2. : contraposée : un chemin augmentant permet d'augmenter f ;
- 2. $\Rightarrow$ 3. : cf. Proposition 1 ;

$\square$

Les points 1 et 2 du théorème prouvent la correction de la méthode de Ford-Fulkerson.

Preuve du théorème *max-flow min-cut*

Théorème 4 (Max-flow min-cut)

[1] Soit f un flot dans un réseau $G = (V, A, c)$ de source s et de puits t. Les conditions suivantes sont équivalentes :

1. *f est un flot maximum ;*
2. *le réseau résiduel G_f ne contient pas de chemin augmentant ;*
3. *il existe une coupe (S, T) pour G telle que $|f| = c(S, T)$.*

Démonstration.

- 1. $\Rightarrow$ 2. : contraposée : un chemin augmentant permet d'augmenter f ;
- 2. $\Rightarrow$ 3. : cf. Proposition 1 ;
- 3. $\Rightarrow$ 1. : découle du Corollaire 3.

$\square$

Les points 1 et 2 du théorème prouvent la correction de la méthode de Ford-Fulkerson.

L'algorithme de Dinitz

- Parmi les algorithmes plus efficaces pour calculer des flots maximum, il y a celui de Dinitz ;

L'algorithme de Dinitz

- Parmi les algorithmes plus efficaces pour calculer des flots maximum, il y a celui de Dinitz ;
- Il réalise en $O(|V|^2|A|)$ ce que Edmonds-Karp fait en $O(|V||A|^2)$ (mieux puisque $|A| \geq |V|$ pour un réseau faiblement connexe) ;

L'algorithme de Dinitz

- Parmi les algorithmes plus efficaces pour calculer des flots maximum, il y a celui de Dinitz ;
- Il réalise en $O(|V|^2|A|)$ ce que Edmonds-Karp fait en $O(|V||A|^2)$ (mieux puisque $|A| \geq |V|$ pour un réseau faiblement connexe) ;
- On peut rabaisser sa complexité à $O(|V||A|\log|V|)$ grâce aux *link-cut trees* ou *dynamic trees* [4].

L'algorithme de Dinitz

- Parmi les algorithmes plus efficaces pour calculer des flots maximum, il y a celui de Dinitz ;
- Il réalise en $O(|V|^2|A|)$ ce que Edmonds-Karp fait en $O(|V||A|^2)$ (mieux puisque $|A| \geq |V|$ pour un réseau faiblement connexe) ;
- On peut rabaisser sa complexité à $O(|V||A| \log |V|)$ grâce aux *link-cut trees* ou *dynamic trees* [4].
- Il est assez similaire à l'algorithme d'Edmonds-Karp ;

Présentation de l'algorithme

L'algorithme de Dinitz suit la méthode de Edmonds et Karp, à deux différences près :

Présentation de l'algorithme

L'algorithme de Dinitz suit la méthode de Edmonds et Karp, à deux différences près :

1. on maintient une structure G_L contenant **tous** les plus courts chemins de s à t dans G_f ;

Présentation de l'algorithme

L'algorithme de Dinitz suit la méthode de Edmonds et Karp, à deux différences près :

1. on maintient une structure G_L contenant **tous** les plus courts chemins de s à t dans G_f ;
2. au lieu d'augmenter le flot sur un seul chemin de s à t dans G_f, on l'augmente sur **tous** les chemins trouvés dans G_L ;

Présentation de l'algorithme

L'algorithme de Dinitz suit la méthode de Edmonds et Karp, à deux différences près :

1. on maintient une structure G_L contenant **tous** les plus courts chemins de s à t dans G_f ;
2. au lieu d'augmenter le flot sur un seul chemin de s à t dans G_f, on l'augmente sur **tous** les chemins trouvés dans G_L ;
3. on met ensuite à jour G_f et G_L ;

Présentation de l'algorithme

L'algorithme de Dinitz suit la méthode de Edmonds et Karp, à deux différences près :

1. on maintient une structure G_L contenant **tous** les plus courts chemins de s à t dans G_f ;
2. au lieu d'augmenter le flot sur un seul chemin de s à t dans G_f, on l'augmente sur **tous** les chemins trouvés dans G_L ;
3. on met ensuite à jour G_f et G_L ;

Comme dans la méthode de Ford-Fulkerson, on s'arrête quand G_f ne permet plus d'augmentation (via G_L) ; le flot trouvé est alors maximum.

Le graphe de parcours en largeur

Définition 5

Soit $G = (V, A)$ un graphe orienté, et $s \in V$. Le **graphe de parcours en largeur de** G **au départ de** s est le graphe orienté G_L défini par :

- V_i = sommets de G à distance i de s ; d = distance du sommet le plus éloigné ;
- $V(G_L) = \cup_{i=0}^{d} V_i$;
- $A(G_L) = \cup_{i=1}^{d} A_i$, où $A_i = \{(u, v) \mid u \in V_{i-1}, v \in V_i\}$.

Attention : distance = nombre d'arcs, **pas** somme des poids.

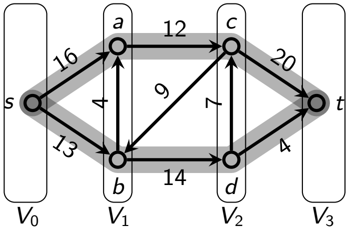
Le graphe de parcours en largeur

Définition 5

Soit $G = (V, A)$ un graphe orienté, et $s \in V$. Le **graphe de parcours en largeur de** G **au départ de** s est le graphe orienté G_L défini par :

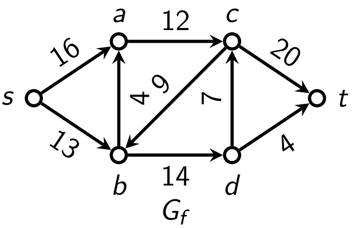
- V_i = sommets de G à distance i de s ; d = distance du sommet le plus éloigné ;
- $V(G_L) = \cup_{i=0}^{d} V_i$;
- $A(G_L) = \cup_{i=1}^{d} A_i$, où $A_i = \{(u, v) \mid u \in V_{i-1}, v \in V_i\}$.

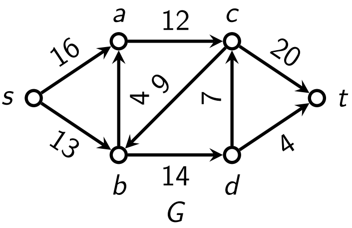
Attention : distance = nombre d'arcs, **pas** somme des poids.



L'algorithme de Dinitz

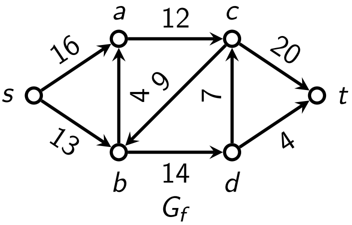
Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.

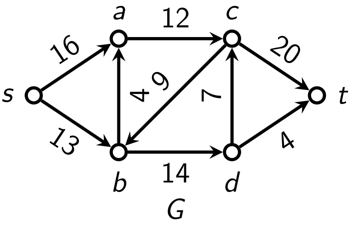


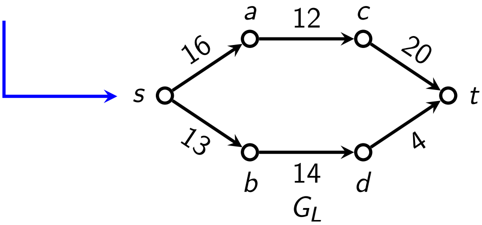


L'algorithme de Dinitz

Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.

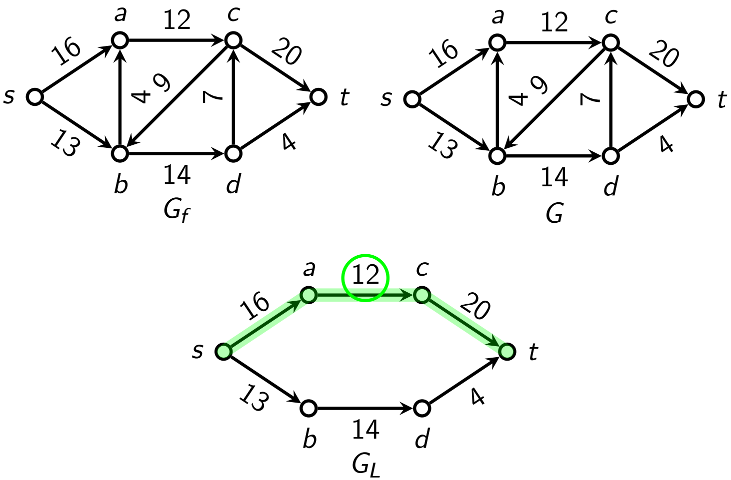






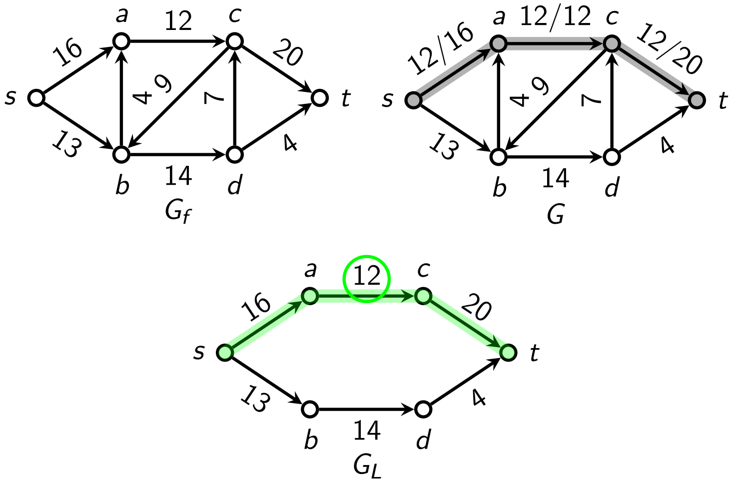
L'algorithme de Dinitz

Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.



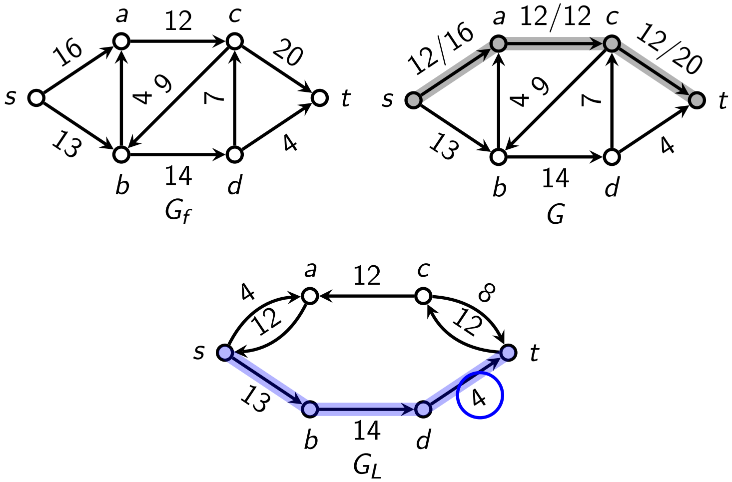
L'algorithme de Dinitz

Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.



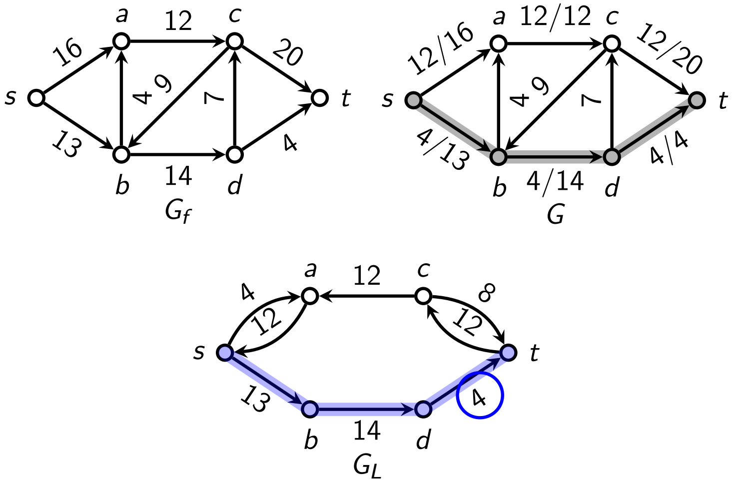
L'algorithme de Dinitz

Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.



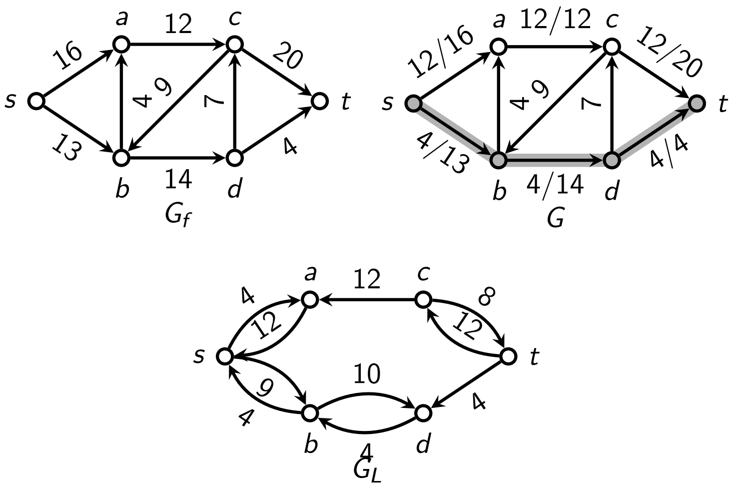
L'algorithme de Dinitz

Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.



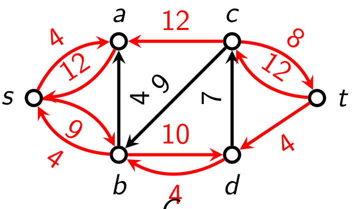
L'algorithme de Dinitz

Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.

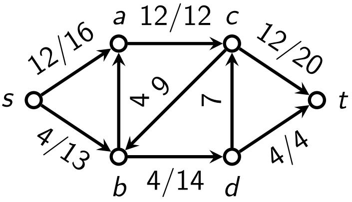


L'algorithme de Dinitz

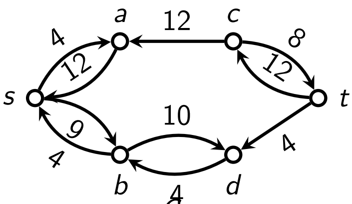
Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.



G_f



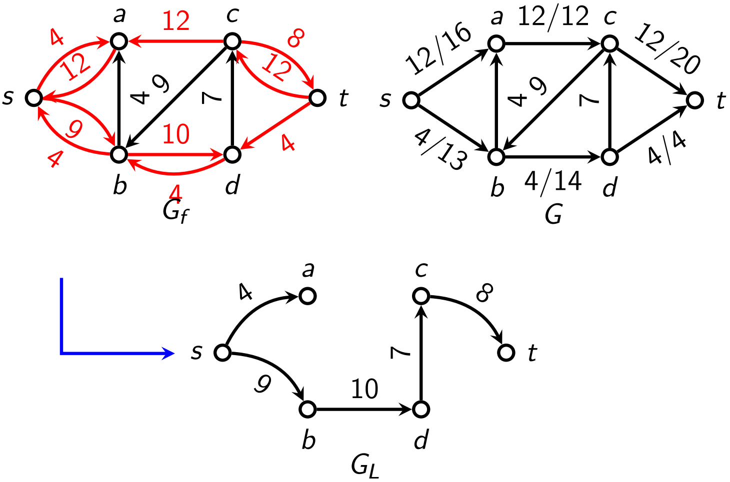
G



G_L

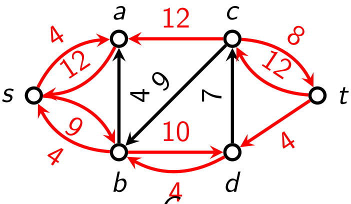
L'algorithme de Dinitz

Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.

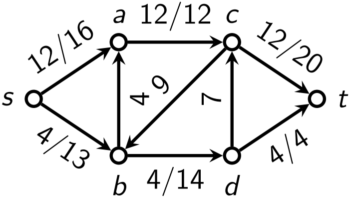


L'algorithme de Dinitz

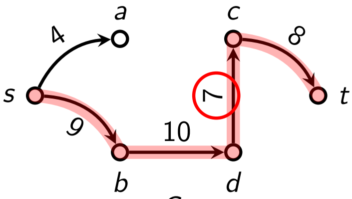
Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.



G_f



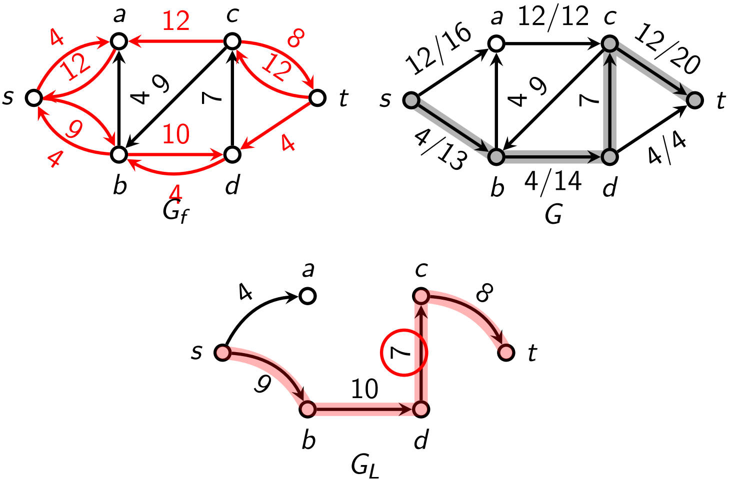
G



G_L

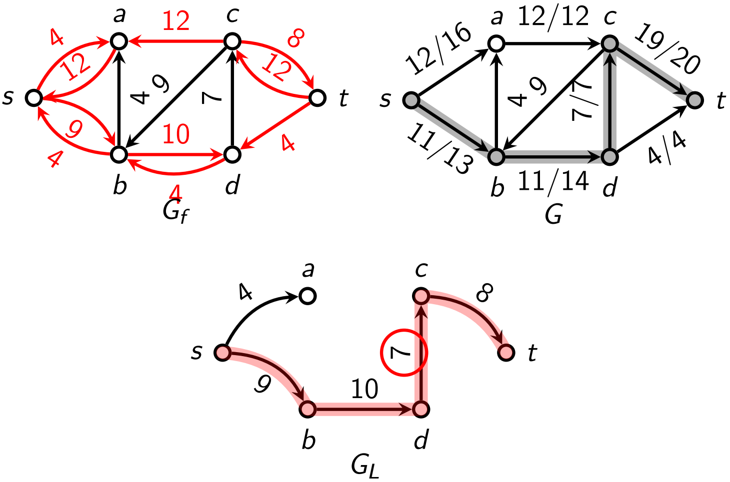
L'algorithme de Dinitz

Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.



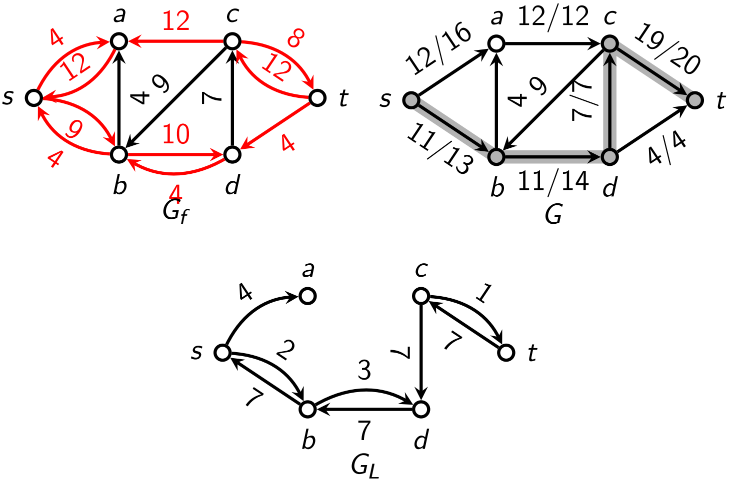
L'algorithme de Dinitz

Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.



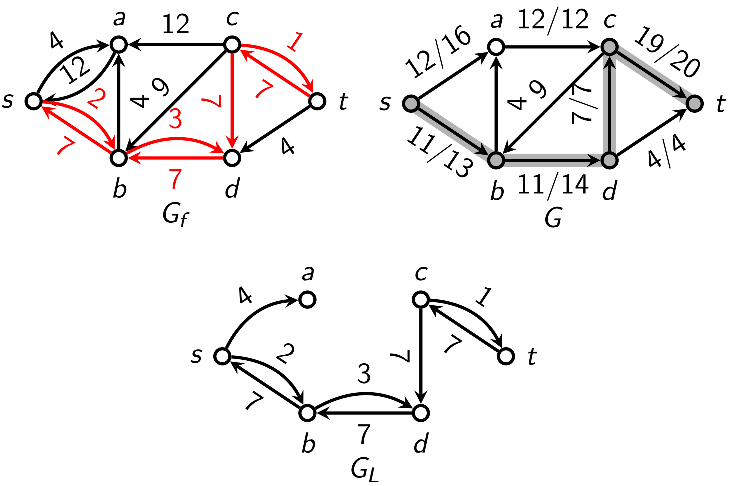
L'algorithme de Dinitz

Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.



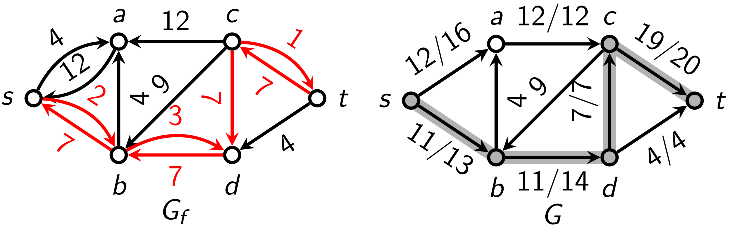
L'algorithme de Dinitz

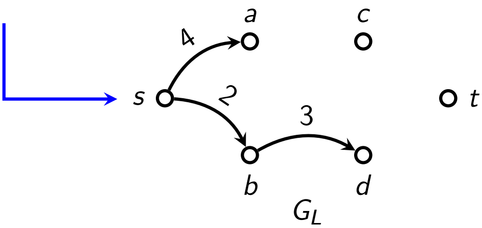
Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.



L'algorithme de Dinitz

Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.





Pseudocode

On peut exprimer l'algorithme de Dinitz en pseudocode comme suit :

Algorithme 1 : $\textsc{Dinitz}(G)$

Entrées : un réseau de flot G.
Sortie : un flot maximum pour G.

flot $\leftarrow$ tableau associatif (clés $=$ G.arcs(), valeurs $= 0$);
source $\leftarrow$ unique sommet de degré entrant nul de G;
puits $\leftarrow$ unique sommet de degré sortant nul de G;
$G_f \leftarrow G$;
$G_L \leftarrow \textsc{DagLargeur}(G$, source, puits$)$;
arcs $\leftarrow \textsc{FlotBloquant}(G_L, f$, source, puits$)$;
tant que *arcs* $\neq \emptyset$ **faire**
  $\textsc{MettreAJourResiduel}(G, G_f$, arcs, flot$)$;
  $G_L \leftarrow \textsc{DagLargeur}(G_f$, source, puits$)$;
  arcs $\leftarrow \textsc{FlotBloquant}(G_L, f$, source, puits$)$;
renvoyer flot;

Flots bloquants

Au lieu d'augmenter le flot sur un chemin dans le résiduel, on calcule un **flot bloquant** dans G_L :

Algorithme 2 : FLOTBLOQUANT(G_L, f, source, puits)

Entrées : un graphe orienté acyclique pondéré G_L, un flot f, une source et un puits.

Résultat : le flot f augmente au maximum le long de chaque chemin de G_L, qui est mis à jour au fur et à mesure ; renvoie l'ensemble des arcs qui ont subi un changement de flot.

arcs $\leftarrow \emptyset$;
chemin $\leftarrow$ CHEMINAUGMENTANT(G_L, source, puits);
tant que *chemin* $\neq$ NIL **faire**
    AUGMENTERFLOT(f, chemin);
    METTREAJOURDAGLARGEUR(G, G_L, chemin.arcs(), f);
    arcs $\leftarrow$ arcs $\cup$ chemin.arcs();
    chemin $\leftarrow$ CHEMINAUGMENTANT(G_L, source, puits);
renvoyer arcs ;

Mise à jour de G_L

Et enfin, il faut mettre à jour tous les plus courts chemins suite aux augmentations de flots (très similaire à MettreAJourResiduel) :

Algorithme 3 : MettreAJourDAGLargeur$(G,\ G_L,\ \text{arcs},\ f)$

Entrées : un réseau de flot G, un graphe orienté acyclique pondéré G_L, un ensemble d'arcs, et un flot f.

Résultat : le poids des arcs spécifiés de G_L est mis à jour sur base de la valeur du flot associé ; les arcs dont le poids devient nul sont supprimés.

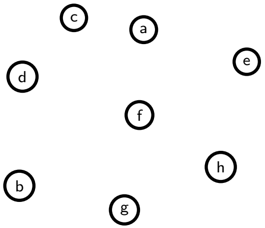
```
pour chaque (u, v, c) ∈ arcs faire
    si G.contient_arc(u, v) alors          // arc original du réseau
        capacité_initiale ← G.poids_arc(u, v) ; flot_actuel ← f[(u, v)];
    sinon                                  // arc inverse du réseau
        capacité_initiale ← G.poids_arc(v, u) ; flot_actuel ← f[(v, u)];
    si flot_actuel = capacité_initiale alors  G_L.supprimer_arc(u, v) ;
    sinon  G_L.ajouter_arc(u, v, capacité_initiale − flot_actuel) ;
```

Conclusions sur Dinitz

- L'approche ressemble à celle de Ford-Fulkerson ;
- L'algorithme est plus efficace (on l'admettra sans preuve) ;
- L'implémentation présentée ici est plus pédagogique, mais on peut faire mieux ;

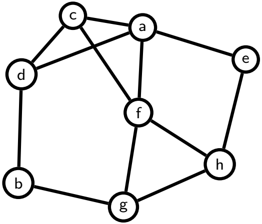
Motivations

- On doit grouper des étudiants en binômes pour un projet ;



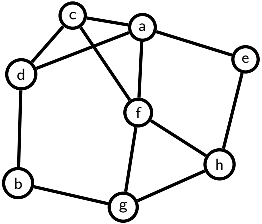
Motivations

- On doit grouper des étudiants en binômes pour un projet ;
- Les étudiants ne sont pas tous amis et communiquent donc leurs préférences, qu'on encode sous la forme d'un graphe ;



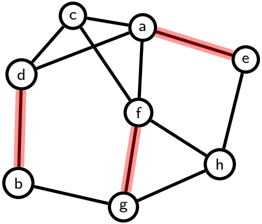
Motivations

- On doit grouper des étudiants en binômes pour un projet ;
- Les étudiants ne sont pas tous amis et communiquent donc leurs préférences, qu'on encode sous la forme d'un graphe ;
- On cherche ensuite à trouver un ensemble d'arêtes disjointes couvrant tous les étudiants :



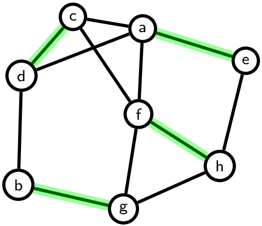
Motivations

- On doit grouper des étudiants en binômes pour un projet ;
- Les étudiants ne sont pas tous amis et communiquent donc leurs préférences, qu'on encode sous la forme d'un graphe ;
- On cherche ensuite à trouver un ensemble d'arêtes disjointes couvrant tous les étudiants :



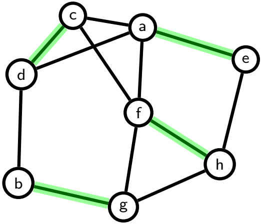
Motivations

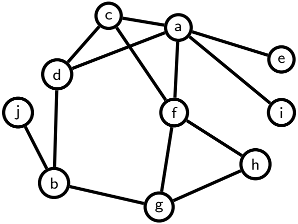
- On doit grouper des étudiants en binômes pour un projet ;
- Les étudiants ne sont pas tous amis et communiquent donc leurs préférences, qu'on encode sous la forme d'un graphe ;
- On cherche ensuite à trouver un ensemble d'arêtes disjointes couvrant tous les étudiants :



Motivations

- On doit grouper des étudiants en binômes pour un projet ;
- Les étudiants ne sont pas tous amis et communiquent donc leurs préférences, qu'on encode sous la forme d'un graphe ;
- On cherche ensuite à trouver un ensemble d'arêtes disjointes couvrant tous les étudiants :

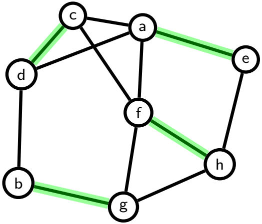


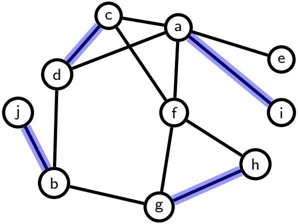


- i et j se sont réveillés un peu tard ; e et h se sont disputés ;

Motivations

- On doit grouper des étudiants en binômes pour un projet ;
- Les étudiants ne sont pas tous amis et communiquent donc leurs préférences, qu'on encode sous la forme d'un graphe ;
- On cherche ensuite à trouver un ensemble d'arêtes disjointes couvrant tous les étudiants :





- *i* et *j* se sont réveillés un peu tard ; *e* et *h* se sont disputés ;

Motivations

- Dans certaines situations plus simples, le graphe $G = (V_1 \cup V_2, E)$ des préférences est biparti ; par exemple :

Motivations

- Dans certaines situations plus simples, le graphe $G = (V_1 \cup V_2, E)$ des préférences est biparti ; par exemple :
  - $V_1 =$ développeurs, $V_2 =$ tâches, $E =$ compétence de chaque développeur pour chaque tâche ;

Motivations

- Dans certaines situations plus simples, le graphe $G = (V_1 \cup V_2, E)$ des préférences est biparti ; par exemple :
    - V_1 = développeurs, V_2 = tâches, E = compétence de chaque développeur pour chaque tâche ;
    - V_1 = entrepreneurs, V_2 = travaux, E = capacité de chaque entrepreneur pour chaque poste ;

Motivations

- Dans certaines situations plus simples, le graphe $G = (V_1 \cup V_2, E)$ des préférences est biparti ; par exemple :
  - $V_1 =$ développeurs, $V_2 =$ tâches, $E =$ compétence de chaque développeur pour chaque tâche ;
  - $V_1 =$ entrepreneurs, $V_2 =$ travaux, $E =$ capacité de chaque entrepreneur pour chaque poste ;

  $\vdots$

Motivations

- Dans certaines situations plus simples, le graphe $G = (V_1 \cup V_2, E)$ des préférences est biparti ; par exemple :
  - V_1 = développeurs, V_2 = tâches, E = compétence de chaque développeur pour chaque tâche ;
  - V_1 = entrepreneurs, V_2 = travaux, E = capacité de chaque entrepreneur pour chaque poste ;

  $\vdots$

- Dans ce cas-là, on peut alors résoudre notre problème à l'aide d'un calcul de flot maximum ;

Motivations

- Dans certaines situations plus simples, le graphe $G = (V_1 \cup V_2, E)$ des préférences est biparti ; par exemple :
  - V_1 = développeurs, V_2 = tâches, E = compétence de chaque développeur pour chaque tâche ;
  - V_1 = entrepreneurs, V_2 = travaux, E = capacité de chaque entrepreneur pour chaque poste ;

  $\vdots$
- Dans ce cas-là, on peut alors résoudre notre problème à l'aide d'un calcul de flot maximum ;
- Dans le cas où le graphe n'est pas biparti, le problème reste soluble en temps polynomial mais l'algorithme est beaucoup plus complexe [2] ;

Motivations

- Dans certaines situations plus simples, le graphe $G = (V_1 \cup V_2, E)$ des préférences est biparti ; par exemple :
  - V_1 = développeurs, V_2 = tâches, E = compétence de chaque développeur pour chaque tâche ;
  - V_1 = entrepreneurs, V_2 = travaux, E = capacité de chaque entrepreneur pour chaque poste ;

  $\vdots$

- Dans ce cas-là, on peut alors résoudre notre problème à l'aide d'un calcul de flot maximum ;
- Dans le cas où le graphe n'est pas biparti, le problème reste soluble en temps polynomial mais l'algorithme est beaucoup plus complexe [2] ;
- Et si l'on veut des trinômes ? Le problème est NP-difficile [3] ;

Couplage

Un **couplage** dans un graphe $G = (V, E)$ est un ensemble $F \subseteq E$ d'arêtes deux à deux disjointes ; il est :

1. **maximal** si on ne peut pas lui rajouter d'arêtes,

Couplage

Un **couplage** dans un graphe $G = (V, E)$ est un ensemble $F \subseteq E$ d'arêtes deux à deux disjointes ; il est :

1. **maximal** si on ne peut pas lui rajouter d'arêtes,
2. **maximum** s'il n'existe aucun autre couplage possédant plus d'arêtes, et

Couplage

Un **couplage** dans un graphe $G = (V, E)$ est un ensemble $F \subseteq E$ d'arêtes deux à deux disjointes ; il est :

1. **maximal** si on ne peut pas lui rajouter d'arêtes,
2. **maximum** s'il n'existe aucun autre couplage possédant plus d'arêtes, et
3. **parfait** s'il couvre tous les sommets du graphe.

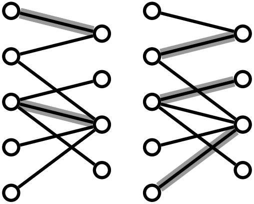
Couplage

Un **couplage** dans un graphe $G = (V, E)$ est un ensemble $F \subseteq E$ d'arêtes deux à deux disjointes ; il est :

1. **maximal** si on ne peut pas lui rajouter d'arêtes,
2. **maximum** s'il n'existe aucun autre couplage possédant plus d'arêtes, et
3. **parfait** s'il couvre tous les sommets du graphe.

Exemple 1

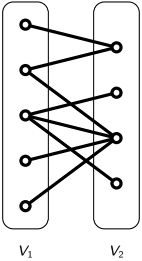
Voici un graphe biparti avec un couplage (a) maximal et (b) maximum.



(a) (b)

Lien entre couplages et flots

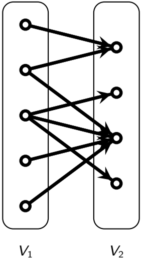
Si $G = (V_1 \cup V_2, E)$ **est biparti**, on peut trouver un couplage maximum à l'aide d'un flot maximum comme suit :



Lien entre couplages et flots

Si $G = (V_1 \cup V_2, E)$ est biparti, on peut trouver un couplage maximum à l'aide d'un flot maximum comme suit :

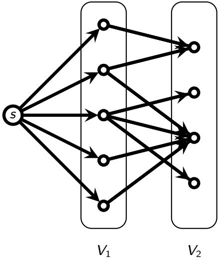
1. orienter toutes les arêtes de G de V_1 vers V_2 ;



Lien entre couplages et flots

Si $G = (V_1 \cup V_2, E)$ est biparti, on peut trouver un couplage maximum à l'aide d'un flot maximum comme suit :

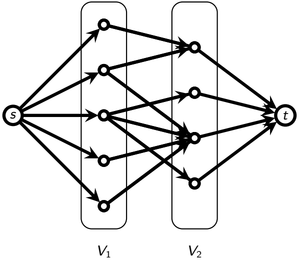
1. orienter toutes les arêtes de G de V_1 vers V_2 ;
2. rajouter une source s avec comme successeurs tout V_1 ;



Lien entre couplages et flots

Si $G = (V_1 \cup V_2, E)$ est biparti, on peut trouver un couplage maximum à l'aide d'un flot maximum comme suit :

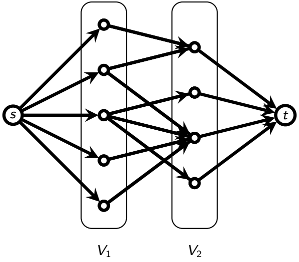
1. orienter toutes les arêtes de G de V_1 vers V_2 ;
2. rajouter une source s avec comme successeurs tout V_1 ;
3. rajouter un puits t avec comme prédécesseurs tout V_2 ;



Lien entre couplages et flots

Si $G = (V_1 \cup V_2, E)$ **est biparti**, on peut trouver un couplage maximum à l'aide d'un flot maximum comme suit :

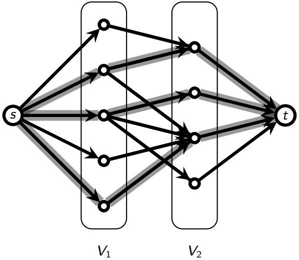
1. orienter toutes les arêtes de G de V_1 vers V_2 ;
2. rajouter une source s avec comme successeurs tout V_1 ;
3. rajouter un puits t avec comme prédécesseurs tout V_2 ;
4. affecter à tous les arcs une capacité de 1 ;



Lien entre couplages et flots

Si $G = (V_1 \cup V_2, E)$ est biparti, on peut trouver un couplage maximum à l'aide d'un flot maximum comme suit :

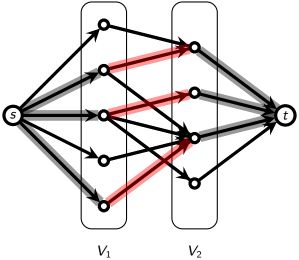
1. orienter toutes les arêtes de G de V_1 vers V_2 ;
2. rajouter une source s avec comme successeurs tout V_1 ;
3. rajouter un puits t avec comme prédécesseurs tout V_2 ;
4. affecter à tous les arcs une capacité de 1 ;
5. trouver un flot maximum dans le réseau résultant.



Lien entre couplages et flots

Si $G = (V_1 \cup V_2, E)$ est biparti, on peut trouver un couplage maximum à l'aide d'un flot maximum comme suit :

1. orienter toutes les arêtes de G de V_1 vers V_2 ;
2. rajouter une source s avec comme successeurs tout V_1 ;
3. rajouter un puits t avec comme prédécesseurs tout V_2 ;
4. affecter à tous les arcs une capacité de 1 ;
5. trouver un flot maximum dans le réseau résultant.



Les arêtes du couplage sont les arêtes de G saturées par le flot.

Correction

Proposition 6

Soit $S \subseteq A$ les arcs auxquels un flot de 1 est affecté après le calcul d'un flot maximum f dans G' ; alors l'ensemble M des arcs de S dont les extrémités sont toutes deux dans G est un couplage maximum pour G.

Correction

Proposition 6

Soit $S \subseteq A$ les arcs auxquels un flot de 1 est affecté après le calcul d'un flot maximum f dans G' ; alors l'ensemble M des arcs de S dont les extrémités sont toutes deux dans G est un couplage maximum pour G.

Démonstration.

On a deux propriétés à prouver :

$\square$

Correction

Proposition 6

Soit $S \subseteq A$ les arcs auxquels un flot de 1 est affecté après le calcul d'un flot maximum f dans G' ; alors l'ensemble M des arcs de S dont les extrémités sont toutes deux dans G est un couplage maximum pour G.

Démonstration.

On a deux propriétés à prouver :

1. M **est un couplage :** la conservation des flots empêche une entrée > 1 et une sortie > 1 :

□

Correction

Proposition 6

Soit $S \subseteq A$ les arcs auxquels un flot de 1 est affecté après le calcul d'un flot maximum f dans G' ; alors l'ensemble M des arcs de S dont les extrémités sont toutes deux dans G est un couplage maximum pour G.

Démonstration.

On a deux propriétés à prouver :

1. M **est un couplage :** la conservation des flots empêche une entrée > 1 et une sortie > 1 :

2. M **est maximum :** par contradiction, s'il existe un couplage M' avec $|M'| > |M|$, alors les arêtes de M' correspondent à des arcs dans G' que l'on peut connecter à s et à t pour obtenir un flot f' de valeur supérieure à f, ce qui contredit l'hypothèse "f est maximum".

□

Complexité

Algorithme 4 : CouplageBiparti(G)

Entrées : un graphe biparti connexe $G = (V_1 \cup V_2, E)$.
Sortie : un couplage maximum pour G.

$H \leftarrow$ GrapheOrientéPondéré();
H.ajouter_sommet(s);
H.ajouter_sommet(t);
pour chaque $\{u, v\} \in G$*.aretes() avec* $u \in V_1$ *et* $v \in V_2$ **faire**
    H.ajouter_arc(s, u, 1) ; H.ajouter_arc(u, v, 1) ; H.ajouter_arc(v, t, 1) ;
flot $\leftarrow$ Dinitz(H);
renvoyer $\{\{u, v\} \mid \{u, v\} \in E$ et flot$[(u, v)] = 1\}$;

- Construction du réseau : $O(|V| + |E|)$

Complexité

Algorithme 4 : $\textsc{CouplageBiparti}(G)$

Entrées : un graphe biparti connexe $G = (V_1 \cup V_2, E)$.
Sortie : un couplage maximum pour G.

```
H ← GrapheOrientéPondéré();
H.ajouter_sommet(s);
H.ajouter_sommet(t);
pour chaque {u, v} ∈ G.aretes() avec u ∈ V1 et v ∈ V2 faire
    H.ajouter_arc(s, u, 1) ; H.ajouter_arc(u, v, 1) ; H.ajouter_arc(v, t, 1) ;
flot ← DINITZ(H);
renvoyer {{u, v} | {u, v} ∈ E et flot[(u, v)] = 1} ;
```

- Construction du réseau : $O(|V| + |E|)$
- La valeur d'un flot maximum f est $O(|V|)$ (couplage parfait)

Complexité

Algorithme 4 : $\textsc{CouplageBiparti}(G)$

Entrées : un graphe biparti connexe $G = (V_1 \cup V_2, E)$.
Sortie : un couplage maximum pour G.

$H \leftarrow$ GrapheOrientéPondéré();
H.ajouter_sommet(s);
H.ajouter_sommet(t);
pour chaque $\{u, v\} \in G$*.aretes() avec* $u \in V_1$ *et* $v \in V_2$ **faire**
  H.ajouter_arc(s, u, 1) ; H.ajouter_arc(u, v, 1) ; H.ajouter_arc(v, t, 1) ;
flot $\leftarrow \textsc{Dinitz}(H)$;
renvoyer $\{\{u, v\} \mid \{u, v\} \in E$ et flot$[(u, v)] = 1\}$;

- Construction du réseau : $O(|V| + |E|)$
- La valeur d'un flot maximum f est $O(|V|)$ (couplage parfait)
- Chaque chemin augmente le flot d'au moins 1 $\Rightarrow$ au plus $O(|V|)$ calculs de chemin

Complexité

Algorithme 4 : $\textsc{CouplageBiparti}(G)$

Entrées : un graphe biparti connexe $G = (V_1 \cup V_2, E)$.
Sortie : un couplage maximum pour G.

$H \leftarrow$ GrapheOrientéPondéré();
H.ajouter_sommet(s);
H.ajouter_sommet(t);
pour chaque $\{u, v\} \in G$*.aretes() avec* $u \in V_1$ *et* $v \in V_2$ **faire**
  H.ajouter_arc(s, u, 1) ; H.ajouter_arc(u, v, 1) ; H.ajouter_arc(v, t, 1) ;
flot $\leftarrow \textsc{Dinitz}(H)$;
renvoyer $\{\{u, v\} \mid \{u, v\} \in E$ et flot$[(u, v)] = 1\}$;

- Construction du réseau : $O(|V| + |E|)$
- La valeur d'un flot maximum f est $O(|V|)$ (couplage parfait)
- Chaque chemin augmente le flot d'au moins 1 $\Rightarrow$ au plus $O(|V|)$ calculs de chemin
- On peut donc obtenir du $O(|V||E|)$; $\textsc{Dinitz}$ donne du $O(\sqrt{|V|}|E|)$;

Flots entiers

- Il est important, pour que l'algorithme fonctionne, que le flot calculé soit entier ;
- Le **théorème d'intégrité**, qu'on admettra sans preuve, garantit son existence ; plus précisément, et plus généralement :

Théorème 7 (Flow integrality theorem)

[1] Soit $G = (V, A, c)$ un réseau de flot avec $c(u, v) \in \mathbb{N}\ \forall\ (u, v) \in A$. Alors le flot f trouvé par la méthode de Ford-Fulkerson satisfait $f(u, v) \in \mathbb{N}\ \forall\ (u, v) \in A$.

Bibliographie

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.
Introduction to Algorithms.
MIT Press, 3ème edition, 2009.

[2] Jack Edmonds.
Paths, trees, and flowers.
Canadian Journal of Mathematics, 17 :449–467, 1965.

[3] M. R. Garey and David S. Johnson.
Computers and Intractability : A Guide to the Theory of NP-Completeness.
W. H. Freeman, 1979.

[4] Daniel Dominic Sleator and Robert Endre Tarjan.
A data structure for dynamic trees.
Journal of Computer and System Sciences, 26(3) :362–391, 1983.