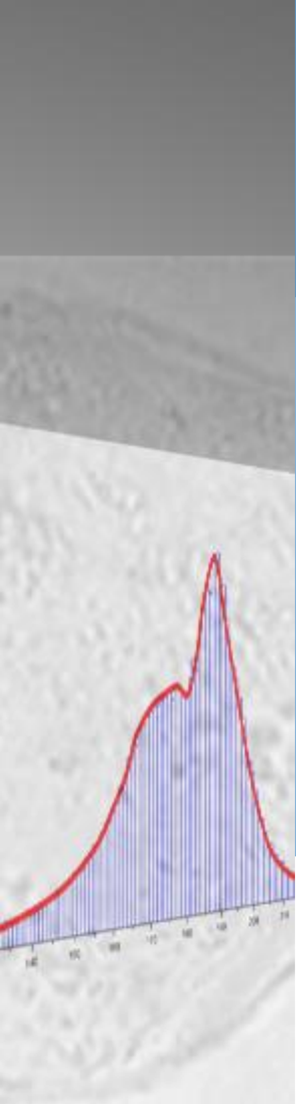


Cours de segmentation d’images

Partie 4 – Graph cuts

Master M2TI - Paris V

2016-2017

John Chaussard

LAGA – Université Paris 13

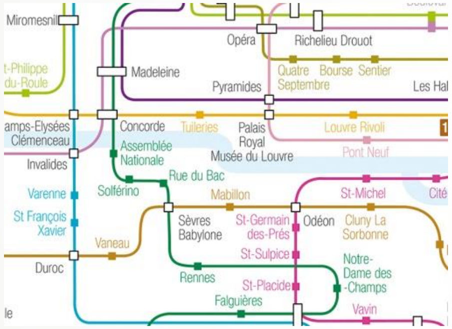
chaussard@math.univ-paris13.fr

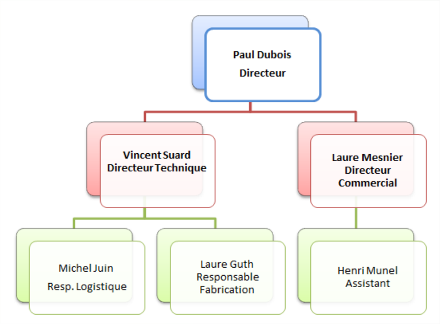
Une très courte histoire des graphes

Une représentation bien pratique

Les graphes sont une structure de données très pratique, présente dans beaucoup d'applications du quotidien...

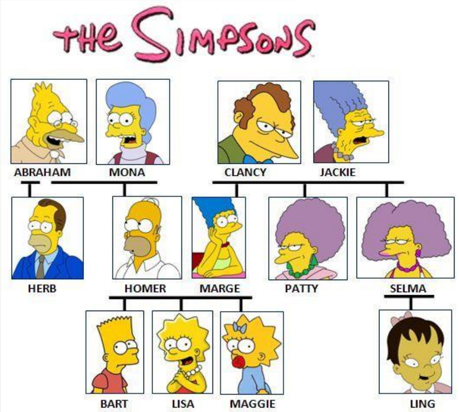
Plans

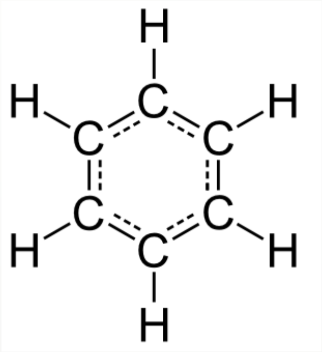




Organigrammes

Arbres généalogiques

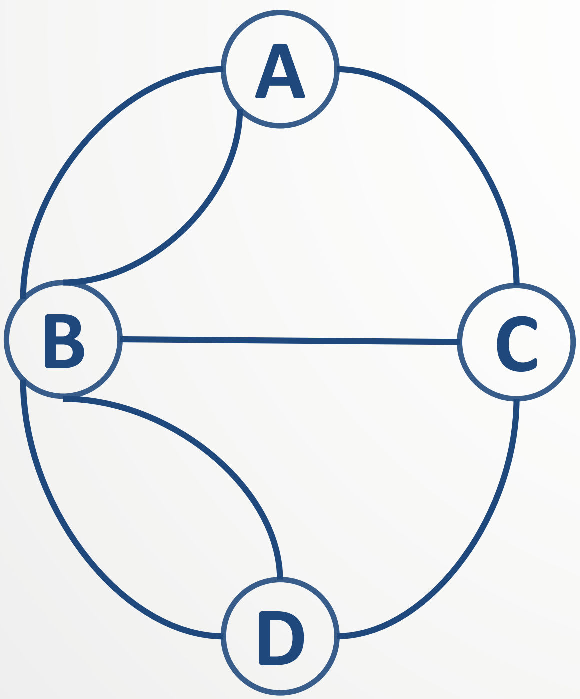


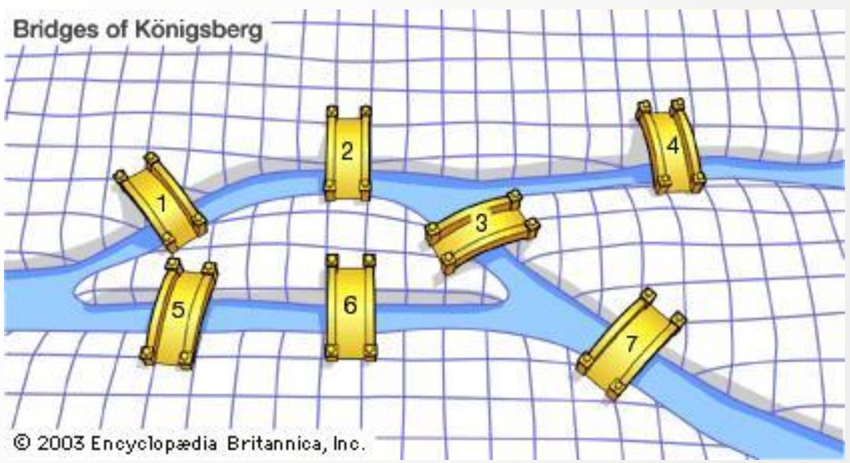


Chimie

Origine des graphes

La représentation par graphe est connue et utilisée depuis bien longtemps, mais **Euler** semble être le premier à avoir formalisé cette structure mathématique pour y résoudre un problème de façon rigoureuse.





Vocabulaire des graphes

Un graphe est constitué :

. D'un ensemble V de **sommets**

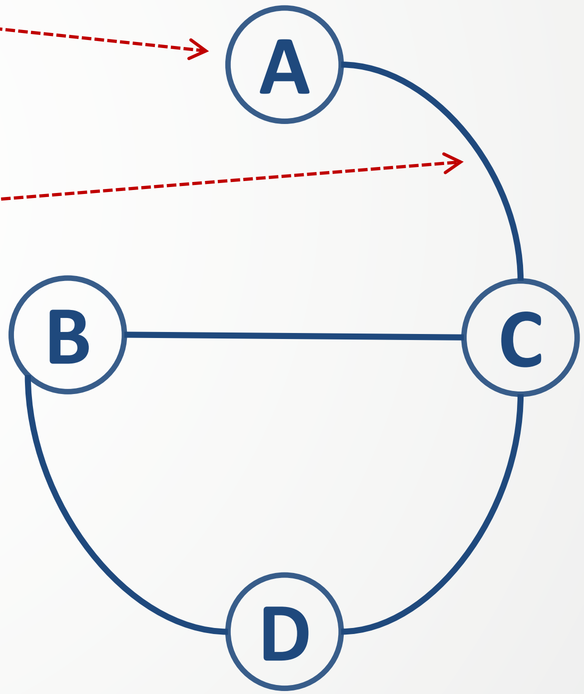
Ici, V={A, B, C, D}

. D'un ensemble E d'**arêtes**

Nous avons ici comme arête

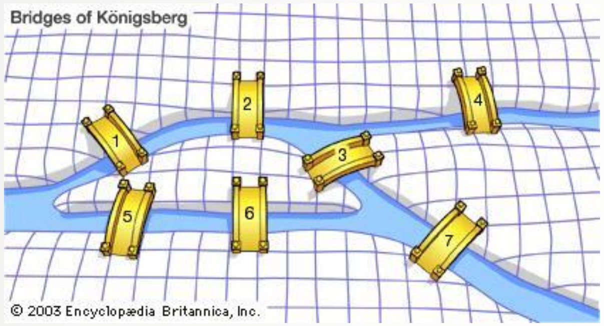
{A, C}, {B, C}, {B, D} et {C, D}

Une arête est en fait une paire de sommets qui sont « liés »

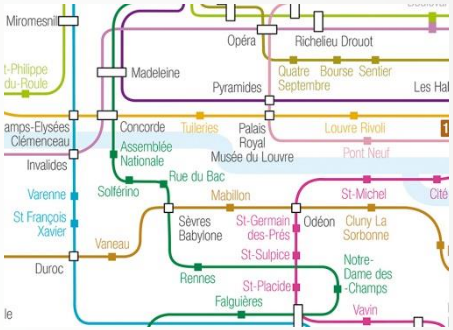


On note en général le **graphe** G=(V,E)

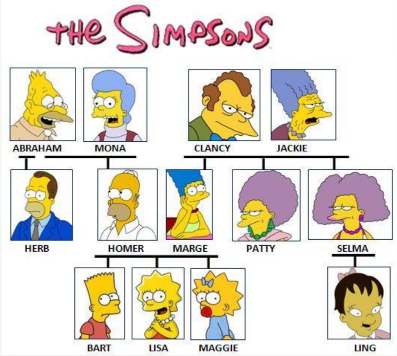
Vocabulaire des graphes



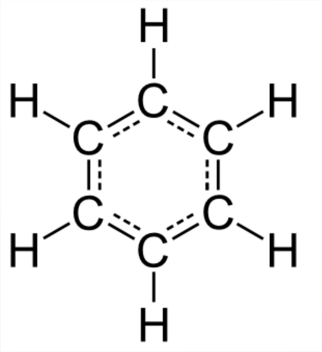
Les sommets sont les rives, les arêtes sont les ponts



Les sommets sont les stations, les arêtes sont les voies reliant les stations



Les sommets sont les personnes, les arêtes sont les relations filiales



Les sommets sont les atomes, les arêtes sont les liaisons atomiques

Résolution de problèmes sur les graphes

Différents problèmes ont été modélisés par des graphes afin d'y trouver des algorithmes permettant d'apporter rapidement une solution :

. Modéliser un **plan** par un graphe, et trouver le plus court chemin entre deux lieux (méthode de Bellman-Ford)

. Modéliser un **projet de construction** par un graphe, et y trouver les parties critiques qui ne doivent pas prendre de retard (méthode PERT)

(projet Polaris estimé à 7 ans, réalisé en 4)

. Modéliser un **réseau routier** par un graphe afin de calculer le débit maximum de voiture qui peut y transiter (méthode de Ford & Fulkerson)

Vocabulaire des graphes (2)

Soit un graphe G=(V,E)

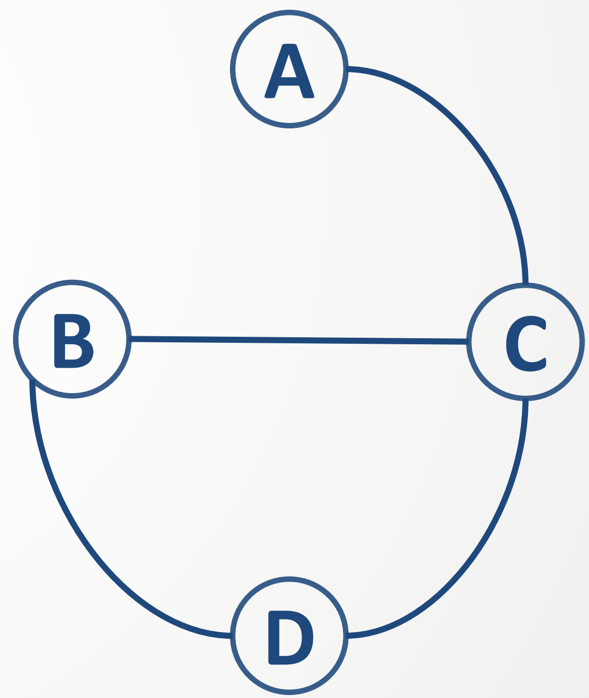
. Deux sommets **sont voisins** s'ils sont reliés par une arête.

ex : A et C sont voisins

ex : D et A ne sont pas voisins

. On note Γ la fonction qui associe à un sommet du graphe l'ensemble de ses voisins

ex : $\Gamma(B) = \{C,D\}$



Débruitage et minimisation

Image binaire bruitée

On se pose le problème suivant :

On possède une image constituée de pixels à 0 (noirs) ou 1 (blancs).

Cette image **a été bruitée**, c'est à dire que certains pixels blancs sont devenus noirs, et inversement…

Peut-on **retrouver l'image d'origine** automatiquement à partir de l'image bruitée (restauration d'images) ?

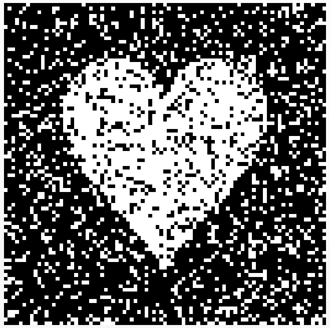


Image binaire bruitée

Pour ce faire, on définit un score sur l'image :

Deux pixels qui se touchent par un bord (on dit qu'ils sont voisins) et qui n'ont pas la même couleur ajouteront 1 au score…

Par exemple :

1	0	1
0	0	0
0	1	1

Score : 6

1	0	0
1	0	0
1	1	1

Score : 4

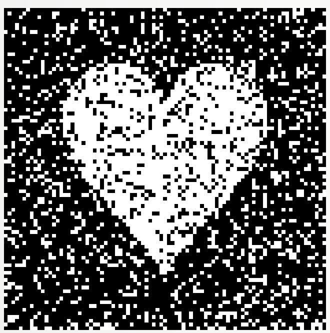
1	1	1
1	1	1
1	1	1

Score : 0

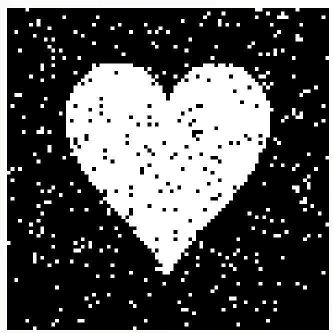
Minimiser le score

Le but sera, à partir de l'image bruitée, de **modifier les valeurs de certains pixels** afin de **minimiser le score** obtenu…

Est-ce que ça marche ?



Score : 4782



Score : 1729

Score : 244

Score : 0

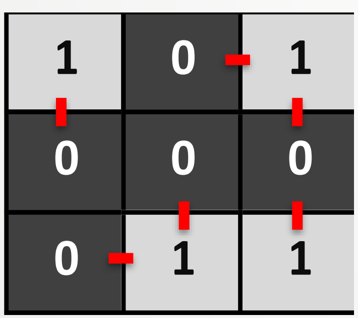
Le score diminue bien au fur et à mesure que l'image devient moins bruitée, mais si on cherchait la solution qui minimise le score, on obtiendrait une image uniforme… **Ca ne fonctionnera pas…**

Image binaire bruitée

On va rajouter un terme au calcul du score afin de décourager la modification des valeurs de pixels…

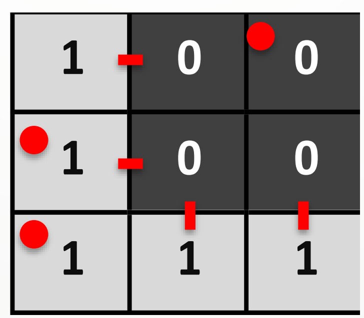
.Un pixel dont la valeur a été modifiée par rapport à l'image originale ajoutera 1 au score.

Par exemple :

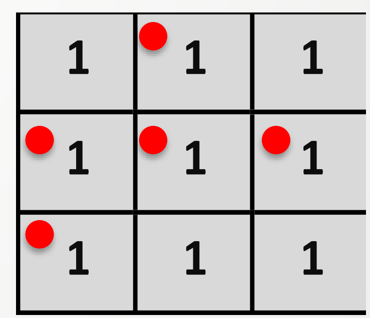


Score : 6

(image originale)



Score : 4+3 = 7

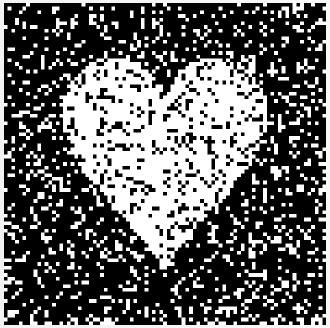


Score : 0+5 = 5

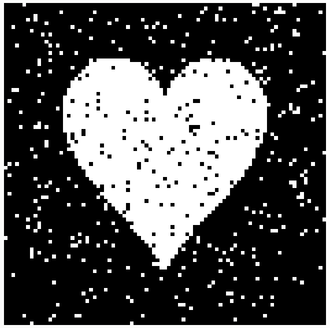
Minimiser le score

Le but du jeu sera, à partir de l'image bruitée, de **modifier les valeurs de certains pixels** afin de **minimiser le score** obtenu…

Est-ce que ça marche ?

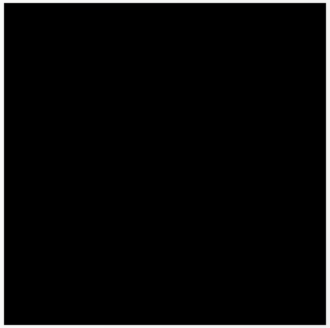


Score : 4782



Score : 3454

Score : 1717



Score : 2728

Le score le plus bas n'est plus obtenu avec l'image uniforme… Le score diminue au fur et à mesure que le bruit diminue : **cette solution devrait fonctionner**.

Minimiser le score

Le but du jeu sera, à partir de l'image bruitée, de **modifier les valeurs de certains pixels** afin de **minimiser le score** obtenu…

On peut écrire le score de façon plus formelle :

Cette partie pénalise les pixels voisins dans l'image qui ne sont pas de même valeur : on l'appelle **le terme de régularisation.**

$$Score(Im, Ref) = \underbrace{\sum_{p \in Im} |Im(p) - Ref(p)|}_{} + \overbrace{\sum_{p,r\ voisins} |Im(p) - Im(r)|}^{}$$

Le score de l'Image par rapport à l'image de Référence

Cette partie pénalise les modifications de valeurs de pixels par rapport à l'image de départ : on l'appelle **le terme d'attache aux données.**

Minimiser le score

Le but du jeu sera, à partir de l'image bruitée, de **modifier les valeurs de certains pixels** afin de **minimiser le score** obtenu…

On peut écrire le score de façon plus formelle :

$$Score(Im, Ref) = \sum_{p \in Im} |Im(p) - Ref(p)| + \sum_{p,r\ voisins} |Im(p) - Im(r)|$$

Ensemble, ces deux termes « se combattent » car, en général, la diminution de l'un entraine l'augmentation de l'autre.

A la fin, **le score minimum est obtenu en faisant le meilleur compromis entre les deux termes**…

Comment minimiser le score ?

Quelle méthode adopter pour, à partir de l'image bruitée, obtenir l'image qui minimise le score vu précédemment ?

Tester toutes les combinaisons de pixels est impossible :

Sur une image de 80 pixels par 80 pixels, il y a en tout 6400 pixels.

Chaque pixel peut prendre deux valeurs (0 ou 1), soit 2^{6400} combinaisons au total…

C'est à peu près un 1 suivi de 1920 zéros…

Sur les supercalculateurs les plus puissants du monde, qui peuvent faire 10^{16} opérations à chaque seconde, cela prendrait 10^{1896} années, c'est à dire 10^{1886} fois l'âge de l'univers…

c'est un peu long…

10^{1896} années

À peu près l'âge de l'univers…

10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000 années

Un problème difficile à résoudre

Pour le moment, nous sommes bloqués…

Il existe des algorithmes, appelés « algorithmes de flot maximal », permettant de résoudre un tout autre problème sur les graphes.

Cependant, nous verrons qu'en **modélisant notre problème différemment, sous forme d'un graphe, nous pouvons utiliser ces algorithmes pour trouver le score minimal**.

Algorithme de Ford & Fulkerson

Graphe de flot

Pour mieux comprendre cette partie, il faut imaginer le graphe comme un réseau de plomberie, où les arcs sont des tuyaux où de l'eau circule.

Nous continuons de considérer des graphes pondérés G=(V, E, p), mais cette fois-ci, nous **considérons que p est le diamètre des tuyaux** : plus p est élevé, et plus grande est la quantité d'eau qui peut circuler dans le tuyau.

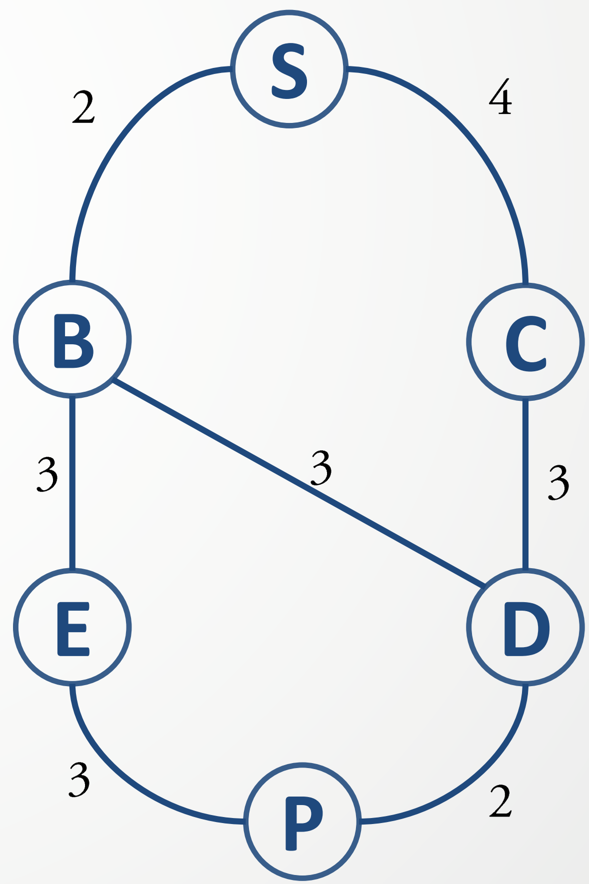
Le problème du flot maximal

On commence avec un graphe pondéré possédant deux sommets spécifiques S et P, appelés **la source et le puits**.

Les poids des arêtes nous indiquent le débit maximal d'eau qui peut circuler dans l'arête.

Par exemple, le tuyau reliant S et C est plus large que la route reliant D à P : il peut y circuler plus d'eau.

Le problème du flot maximal consiste à déterminer le débit maximal d'eau qui peut circuler dans le graphe **entre S (le point de départ) et P (le point d'arrivée).**



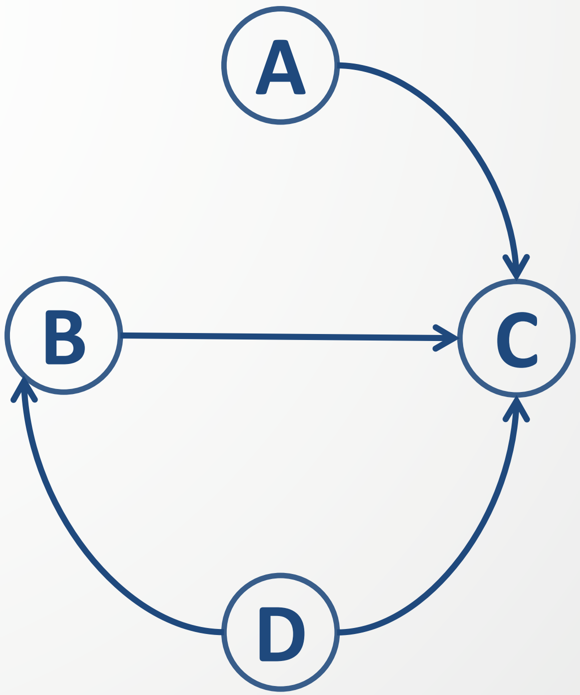
Vocabulaire des graphes (3)

Un graphe orienté est un ensemble de sommets E reliés par des arêtes orientées (on peut les emprunter dans un sens, mais pas dans l'autre) que l'on appelle des **arcs**.

Comme avant, on appelle Γ la fonction qui associe à un sommet du graphe l'ensemble de ses voisins

ex : $\Gamma(B) = \{C\}$

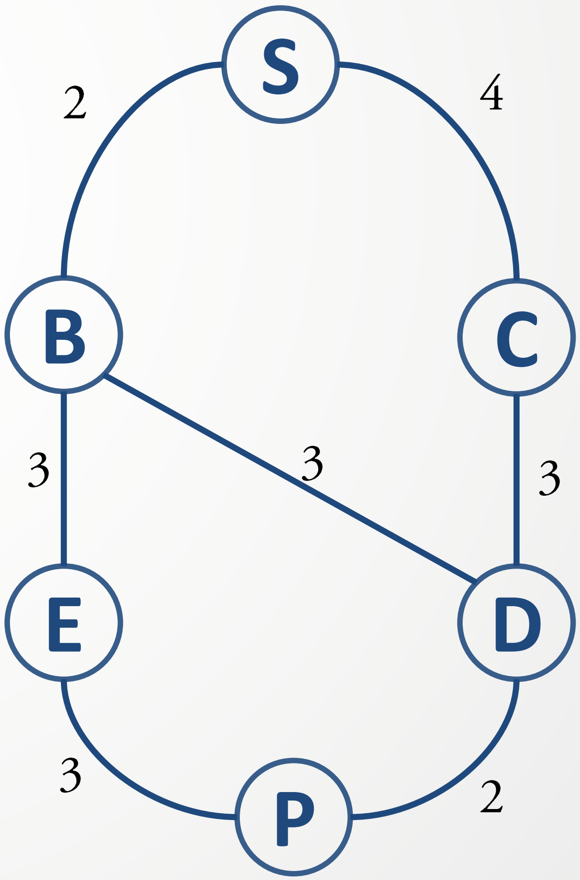
ex : $\Gamma(C) = \emptyset$



Le problème du flot maximal

On commence avec un graphe pondéré possédant deux sommets spécifiques S et P, appelés **la source et le puits**.

On le transforme en un graphe orienté en dédoublant chaque arête en deux arcs pointant dans des sens contraires.



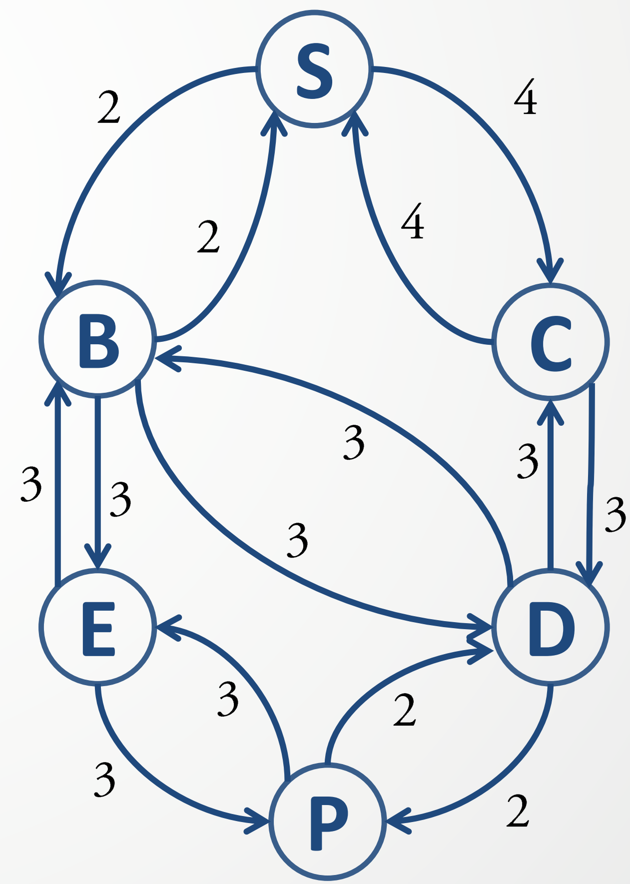
Le problème du flot maximal

On commence avec un graphe pondéré possédant deux sommets spécifiques S et P, appelés **la source et le puits**.

On le transforme en un graphe orienté en dédoublant chaque arête en deux arcs pointant dans des sens contraires.

Le poids de chaque arc est égal au poids de l'arête lui ayant donné naissance.

On supprime les arcs pointant vers la source, ainsi que les arcs ayant pour origine le puits.



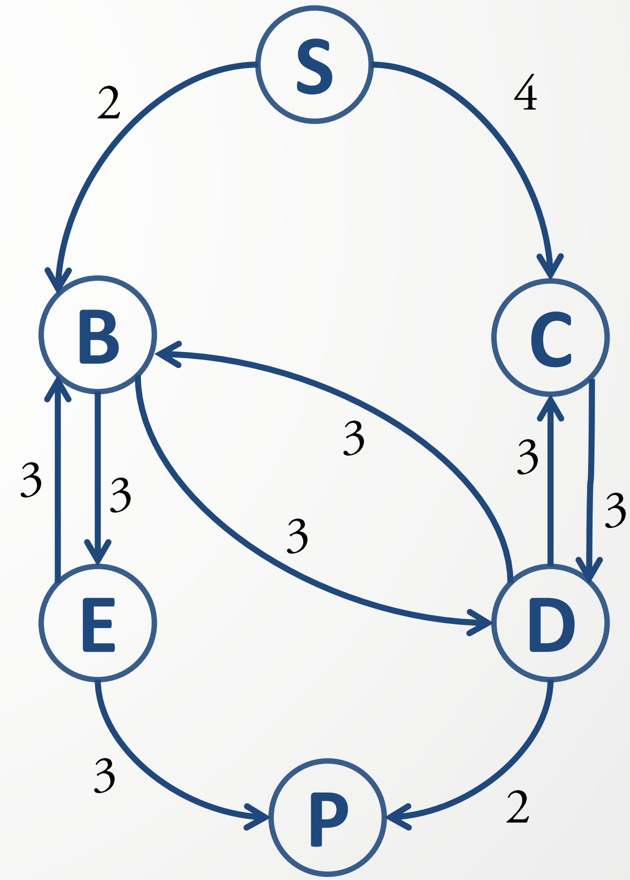
Le problème du flot maximal

On commence avec un graphe pondéré possédant deux sommets spécifiques S et P, appelés **la source et le puits**.

On le transforme en un graphe orienté en dédoublant chaque arête en deux arcs pointant dans des sens contraires.

Le poids de chaque arc est égal au poids de l'arête lui ayant donné naissance.

On supprime les arcs pointant vers la source, ainsi que les arcs ayant pour origine le puits.



L'algorithme de Ford & Fulkerson

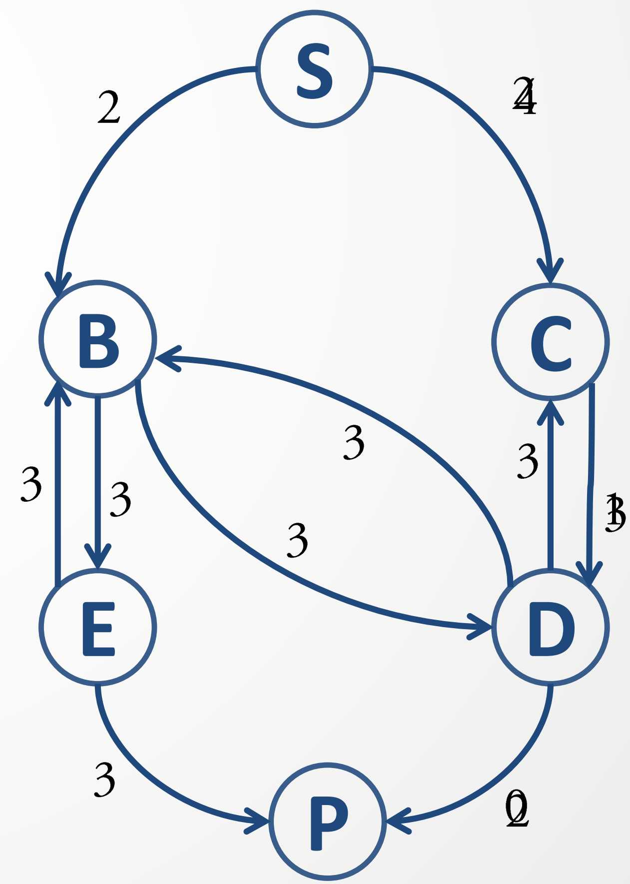
On considère le graphe orienté G=(V,E,p) ainsi que S et P.

On cherche un chemin *che* reliant S à P, ne passant pas par un arc de poids nul.

On trouve *che* = {(S,C), (C,D), (D,P)}

Sur cet ensemble de tuyaux, on peut faire circuler au maximum deux unités d'eau.

On fait donc passer deux unités d'eau à travers ces tuyaux, ce qui a pour effet de diminuer le poids de chaque arc de ce chemin car moins d'eau pourra y circuler à l'avenir (étant donné que deux unités d'eau y circulent déjà).



L'algorithme de Ford & Fulkerson

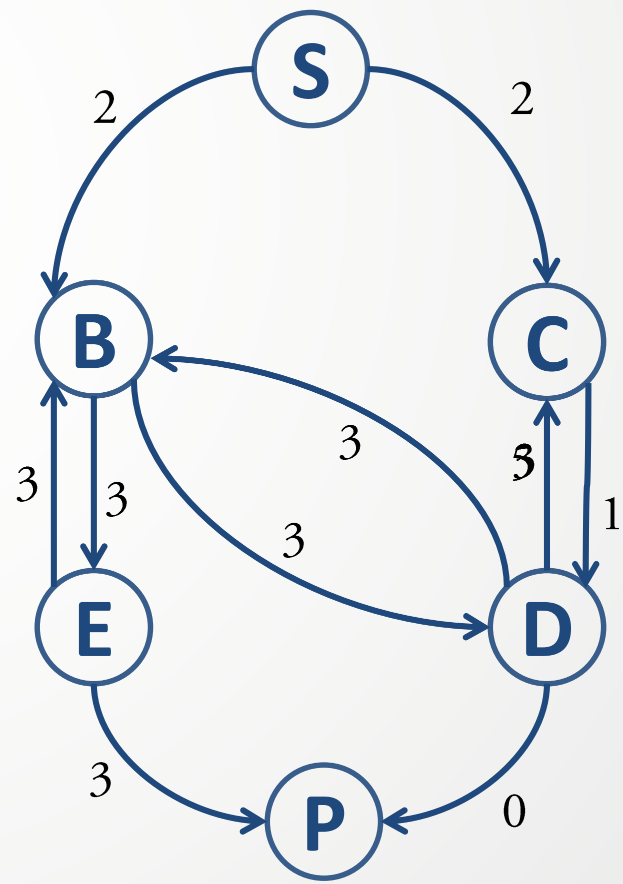
On considère le graphe orienté G=(V,E,p) ainsi que S et P.

On cherche un chemin *che* reliant S à P, ne passant pas par un arc de poids nul.

On trouve *che* = {(S,C), (C,D), (D,P)}

Inversement, on augmente de deux le poids de chaque arc inverse d'un des arcs que l'on vient de modifier…

L'arc (C,S) n'existe pas, ni l'arc (P,D).
L'arc (D,C) existe, son poids passe à 5.



L'algorithme de Ford & Fulkerson

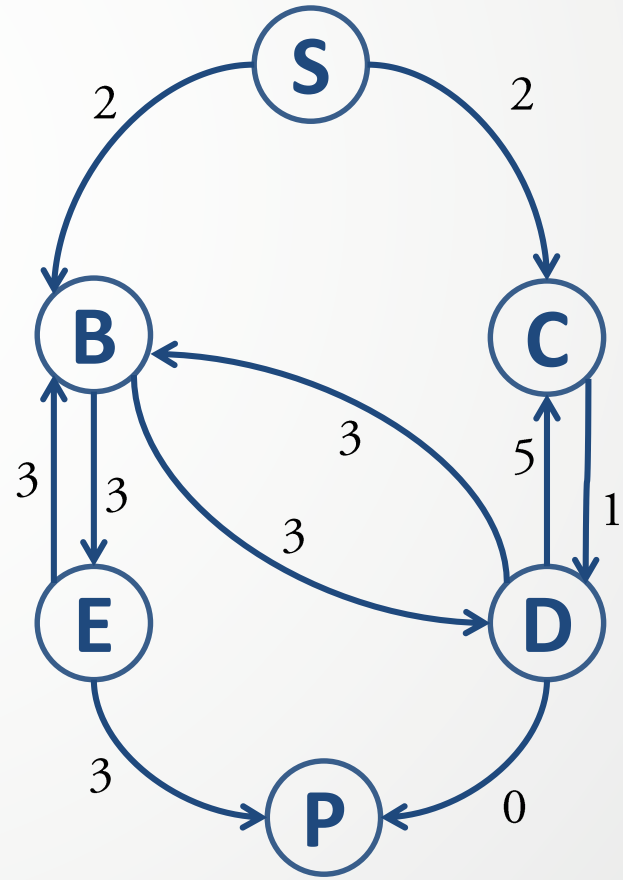
Le poids de l'arc (D,C) passe à 5 ? Qu'est-ce que cela signifie ?

Le tuyau entre C et D peut accueillir 3 unités d'eau en tout… **Pourquoi dire que 5 unités d'eau peuvent circuler de D à C ?**

En réalité, 2 unités d'eau circulent déjà de C vers D…

On peut faire **rebrousser** chemin (de D vers C) à ces 2 unités d'eau,
Puis faire circuler 3 unités d'eau de D vers C…

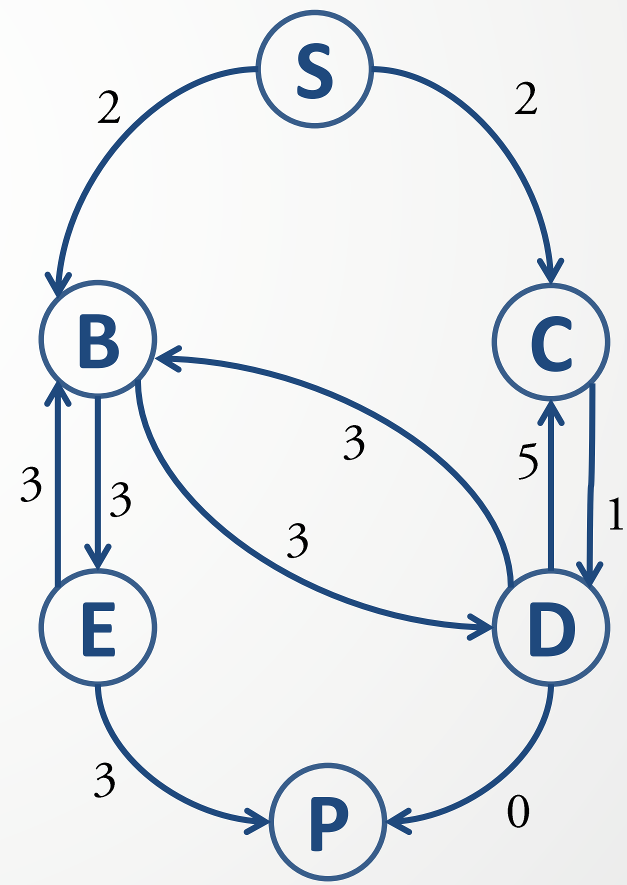
En tout, on aura bien fait circuler 2+3=5 unités d'eau de D vers C…



L'algorithme de Ford & Fulkerson

On considère le graphe orienté G=(V,E,p) ainsi que S et P.

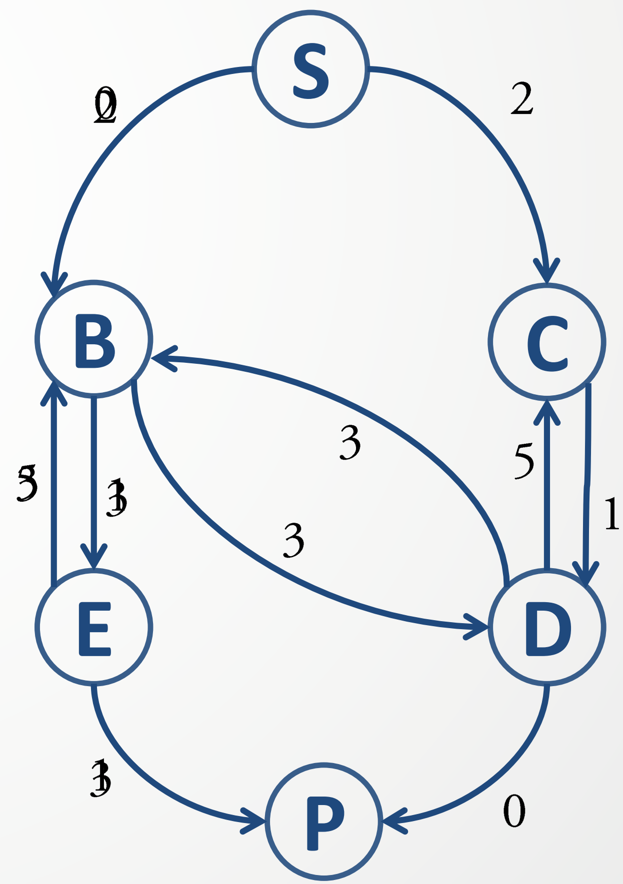
On recommence ces étapes jusqu'à ne plus pouvoir trouver de chemin entre la source et le puit.



L'algorithme de Ford & Fulkerson

On considère le graphe orienté G=(V,E,p) ainsi que S et P.

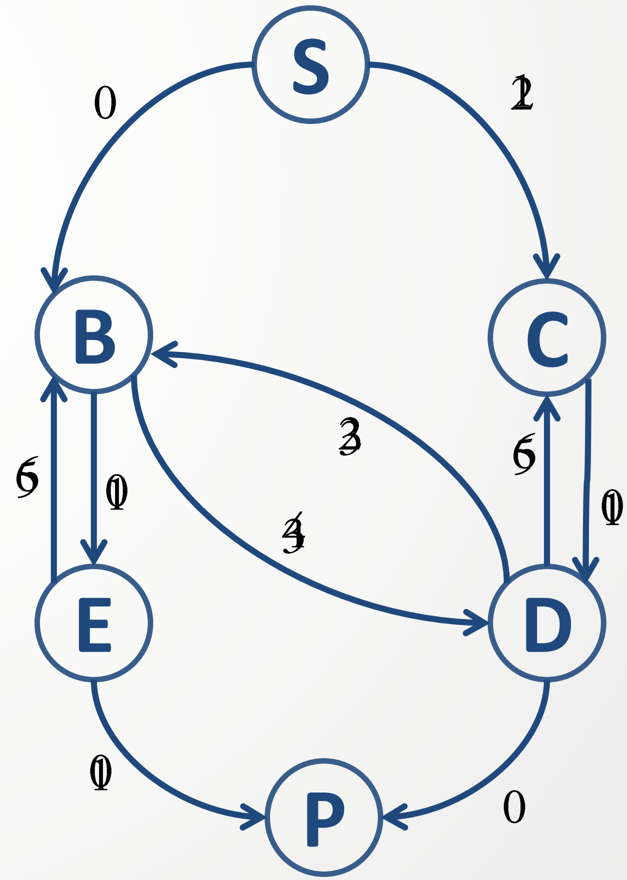
On recommence ces étapes jusqu'à ne plus pouvoir trouver de chemin entre la source et le puit.



L'algorithme de Ford & Fulkerson

On considère le graphe orienté G=(V,E,p) ainsi que S et P.

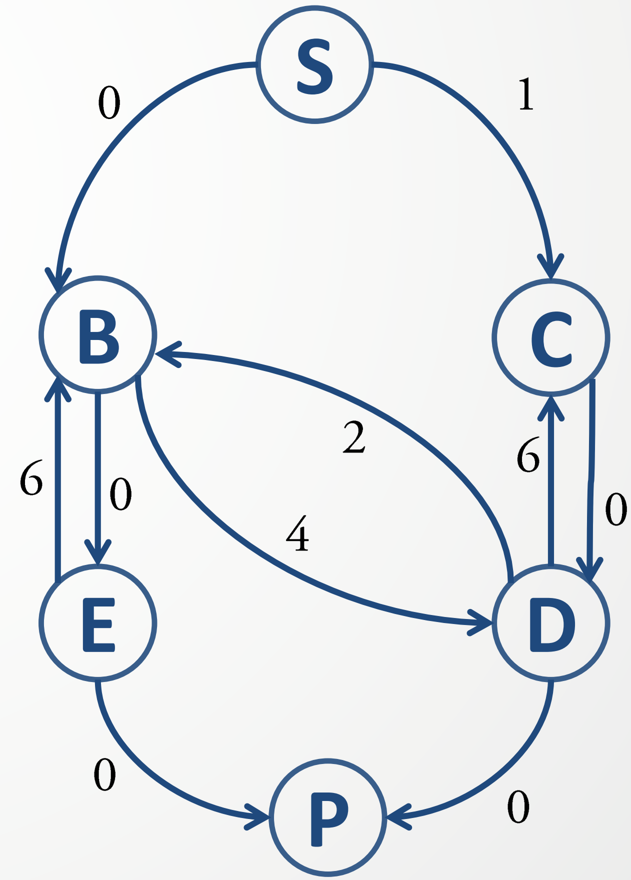
On recommence ces étapes jusqu'à ne plus pouvoir trouver de chemin entre la source et le puit.



Résultat

On considère le graphe orienté G=(V,E,p) ainsi que S et P.

A la fin, on obtient le résultat ci-contre.



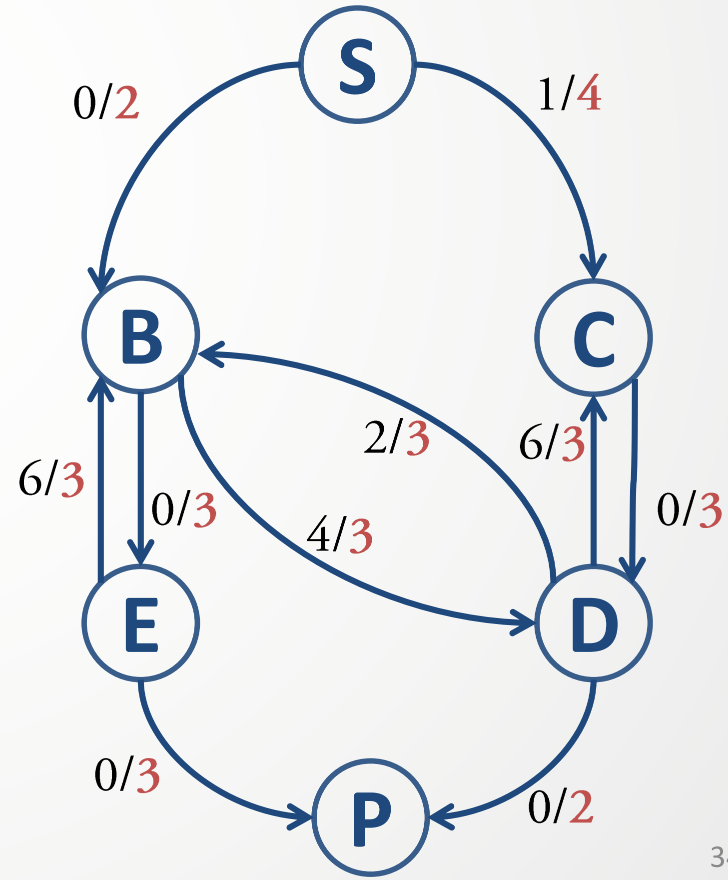
L'algorithme de Ford & Fulkerson

On considère le graphe orienté G=(V,E,p) ainsi que S et P.

A la fin, on obtient le résultat ci-contre.

Les valeurs en rouges sont les capacités originales des routes, tandis que les valeurs en noir sont la quantité d'eau qui peut encore circuler dans chaque tuyau.

On obtient le débit d'eau circulant dans chaque tuyau en soustrayant le nombre en noir du nombre en rouge…



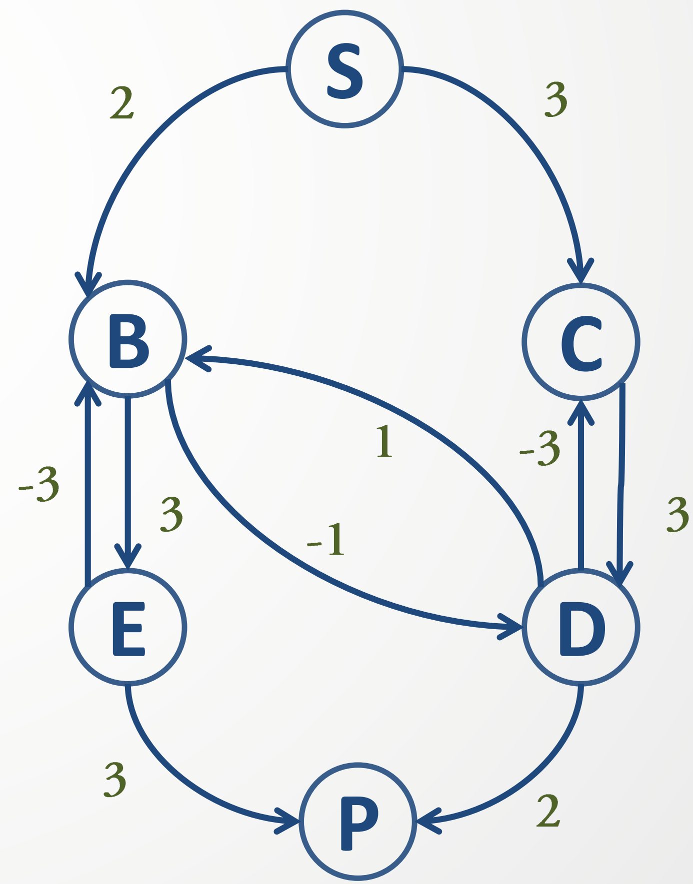
L'algorithme de Ford & Fulkerson

On considère le graphe orienté G=(V,E,p) ainsi que S et P.

A la fin, on obtient le résultat ci-contre.

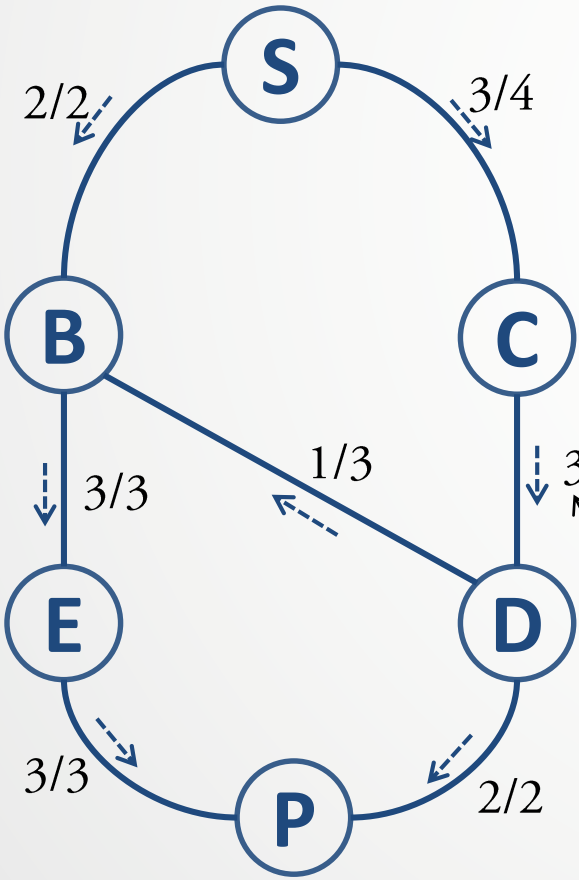
Les valeurs en rouges sont les capacités originales des routes, tandis que les valeurs en noir sont la quantité d'eau qui peut encore circuler dans chaque tuyau.

On obtient le débit d'eau circulant dans chaque tuyau en soustrayant le nombre en noir du nombre en rouge…



L'algorithme de Ford & Fulkerson

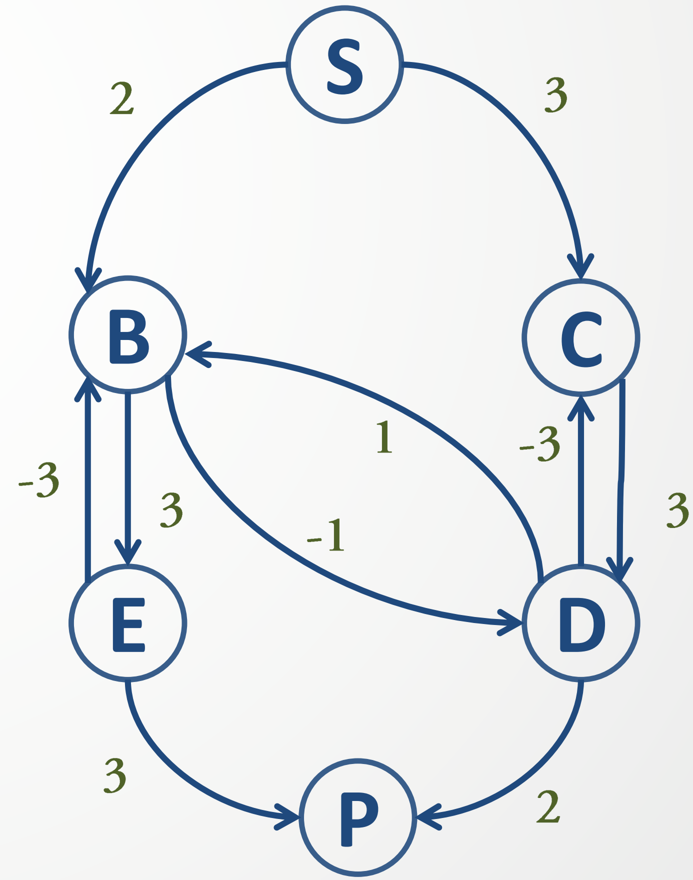
On considère le graphe orienté G=(V,E,p) ainsi que S et P.



On transpose les valeurs obtenues dans le graphe originel en y recopiant uniquement les valeurs positives.

La capacité maximale de l'arête (C,D)

Le débit d'eau circulant sur l'arête (C,D)



L'algorithme de Ford & Fulkerson

On considère le graphe orienté G=(V,E,p) ainsi que S et P.

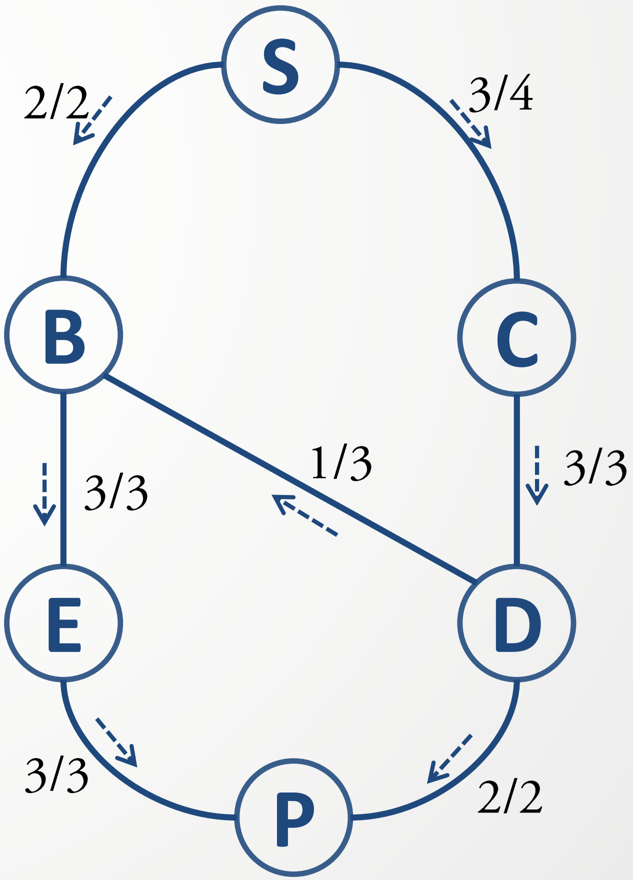
On peut vérifier sur le résultat que :

.Aucun tuyau ne reçoit plus d'eau que sa capacité maximale,

.Toute l'eau entrant par un sommet en ressort,

.La quantité d'eau sortant de S est égale à la quantité d'eau entrant dans P.

En sommant la quantité d'eau partant de S, nous obtenons le flot maximal pouvant circuler entre S et P.



Code de l'algorithme

Voici comment construire le graphe orienté à partir du graphe G :

Soit G=(V,E,p) un graphe pondéré, et deux sommets S et P.

V'=V
E'=∅
Pour chaque arête {A,B} ∈ E,
    Si A≠P et B≠S
        E'= E' ∪ {(A,B)}
        p'((A,B)) = p({A,B})

    Si B≠P et A≠S
        E'= E' ∪ {(B,A)}
        p'((B,A)) = p({A,B})

On construit le graphe orienté pondéré G'(V', E', p').

Code de l'algorithme

Une fois le graphe orienté pondéré construit à partir de G, on réalise l'algorithme de Ford et Fulkerson

Soit G'=(V',E',p') un graphe pondéré orienté, et deux sommets S et P.

Tant qu'il existe un chemin orienté *che* entre S et P ne passant pas par un arc de poids nul

m = $\underset{(A,B)\in che}{\operatorname{argmin}} p'((A,B))$

Pour chaque arc (A,B) ∈ *che*

p'((A,B)) = p'((A,B)) – m

Si A≠S et B≠P

p'((B,A)) = p'((B,A)) + m

Chercher un chemin

L'algorithme stipule « *Tant qu'il existe un chemin orienté che entre S et P ne passant pas par un arc de poids nul* »

Comment trouver un tel chemin ?

On inverse les poids de chaque arc du graphe (les arcs à 0 se retrouveront alors à l'infini), puis on utilise Bellman-Ford (ou tout autre algorithme de plus court chemin) pour trouver un plus court chemin entre S et P.

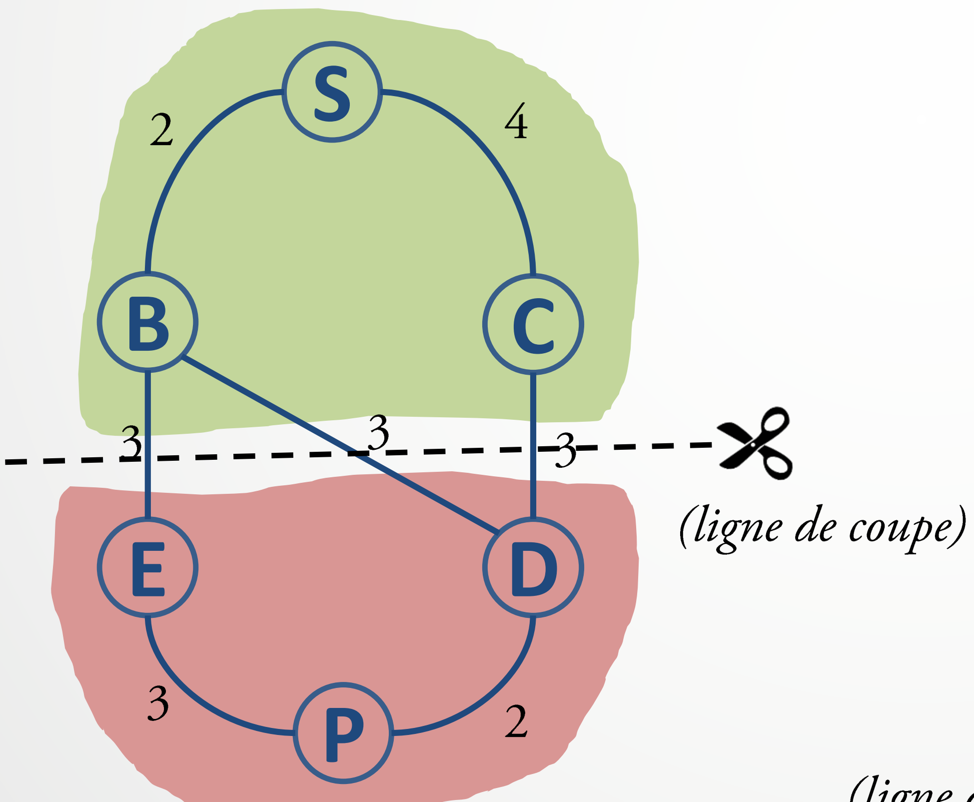
Le plus court chemin nous donnera le chemin pouvant accueillir le plus d'eau entre S et P.

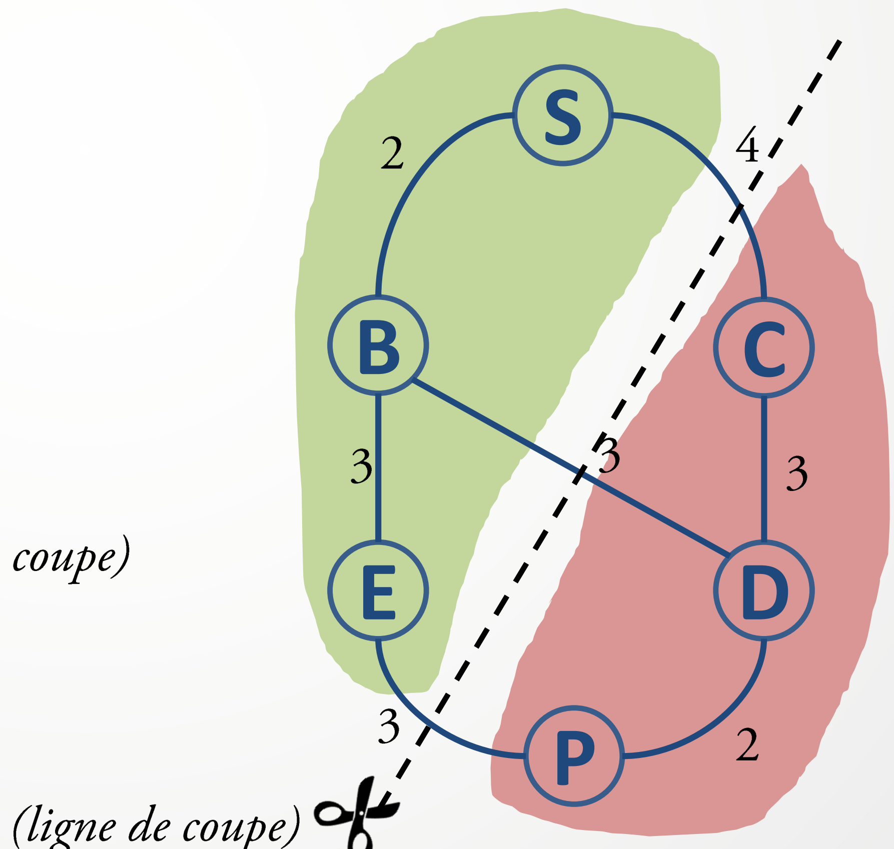
La coupe minimale et les flots

Le problème de la coupe minimale

Etant donné un graphe G pondéré, et deux sommets S et P.

On souhaite couper le graphe (en retirant des arêtes) en deux morceaux : d'un côté le sommet S (côté vert), et de l'autre le sommet P (côté rouge).

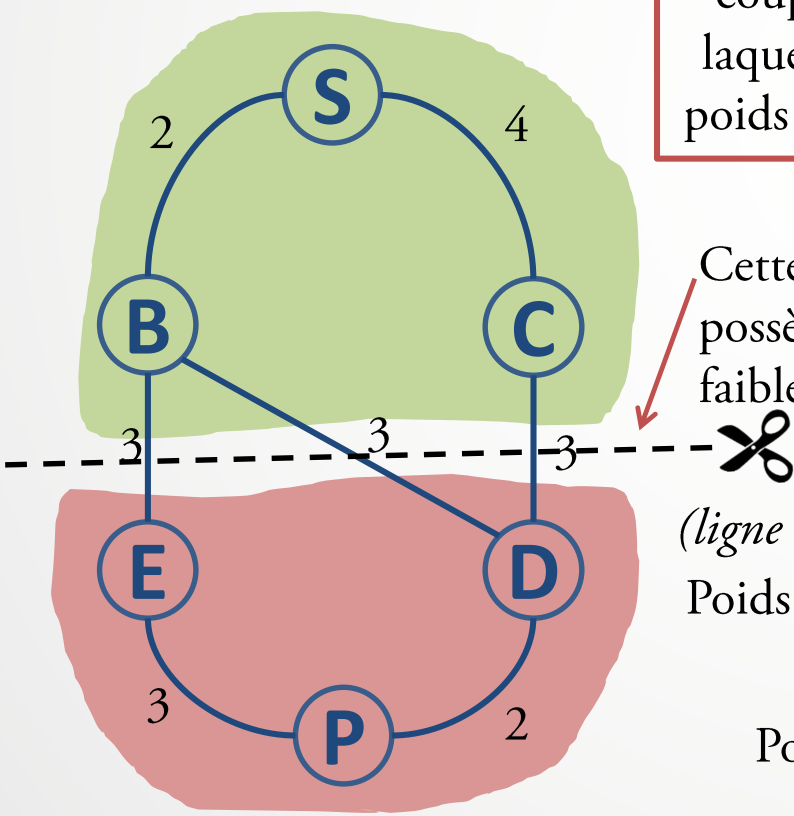




Le problème de la coupe minimale

Le problème de la coupe minimale consiste à trouver la ligne de coupe qui coupera l’ensemble d’arêtes avec le poids le plus faible possible…

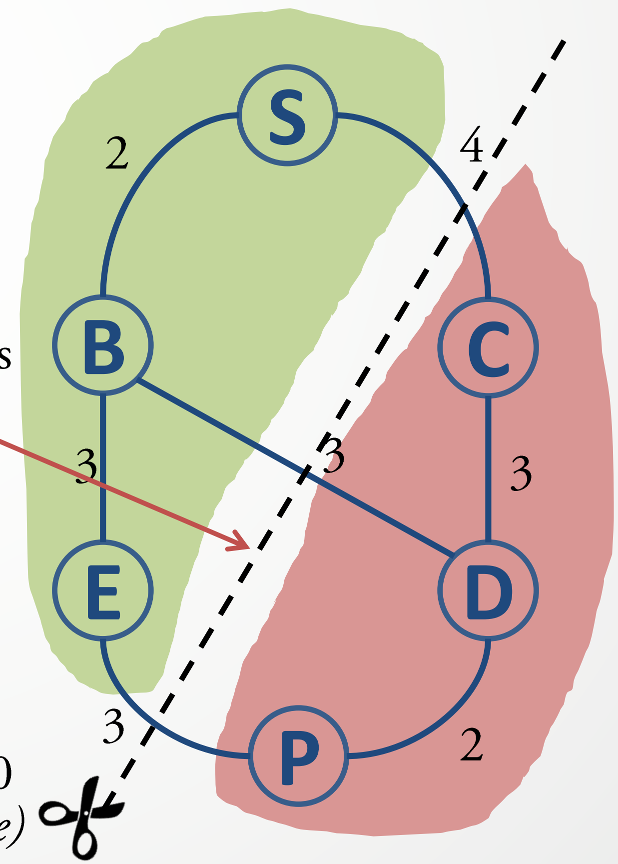
Parmi toutes les coupes possibles, laquelle possède le poids le plus faible ?



Cette ligne de coupe possède un poids plus faible que celle-ci

(ligne de coupe)

Poids : 3+3+3 = 9



Poids : 3+3+4 = 10

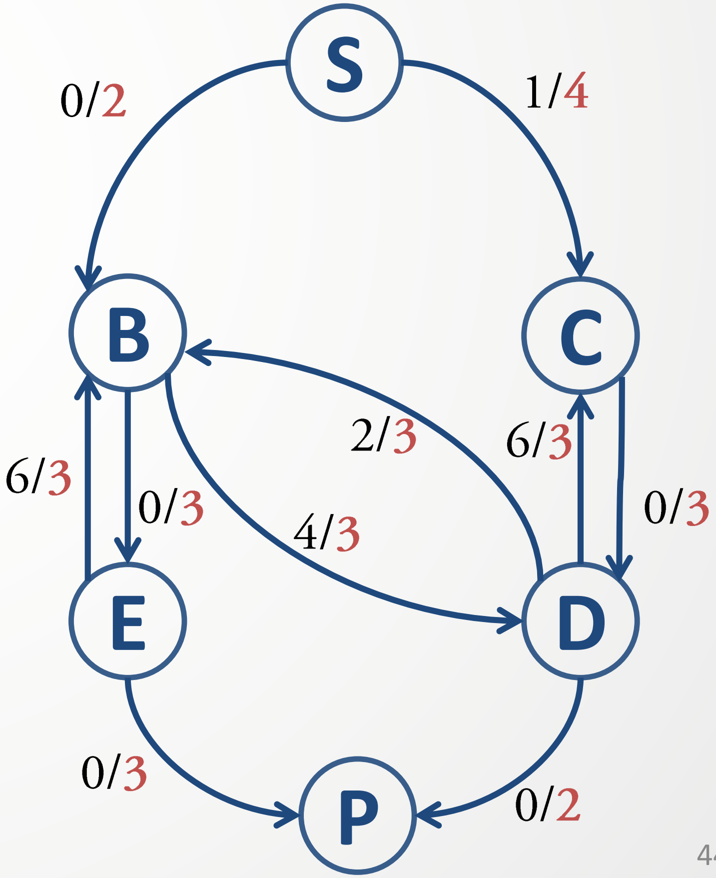
(ligne de coupe)

Coupe minimale et flot

Le problème de la coupe minimale est intimement lié au **problème du flot maximal** dans un graphe…

On regarde chaque arc dont la valeur de capacité restante (le chiffre en noir) est égal à 0…

et on les supprime.



Coupe minimale et flot

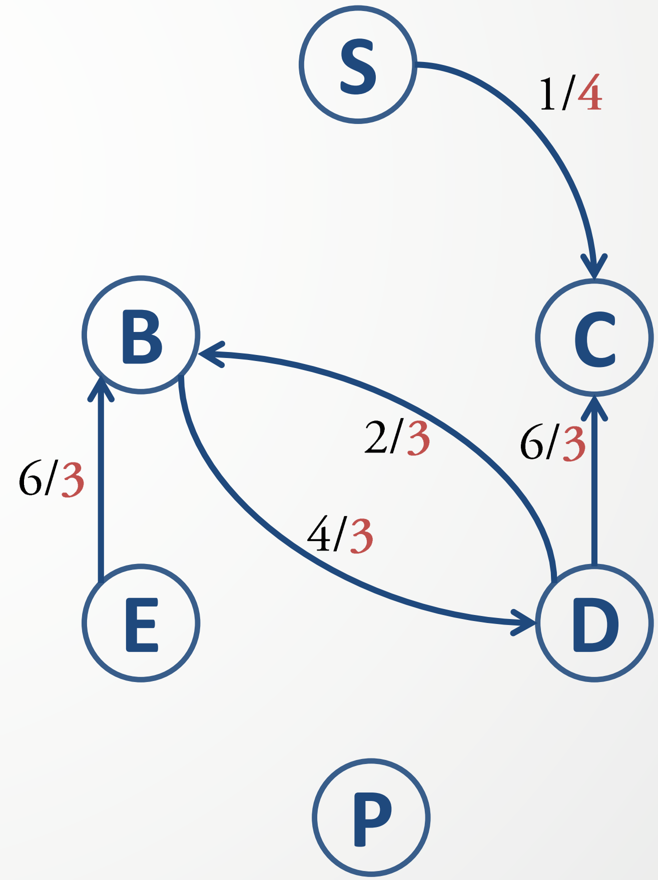
Le problème de la coupe minimale est intimement lié au **problème du flot maximal** dans un graphe…

On regarde chaque arc dont la valeur de capacité restante (le chiffre en noir) est égal à 0…

et on les supprime.

On regarde ensuite quels sommets sont reliés à la source par un chemin (attention, pas le droit d'emprunter les arcs en sens inverse)…

Ici, seul le sommet C est relié à la source.
Ces sommets seront dans le groupe associé à la source, les autre seront dans le groupe du puits.



Coupe minimale et flot

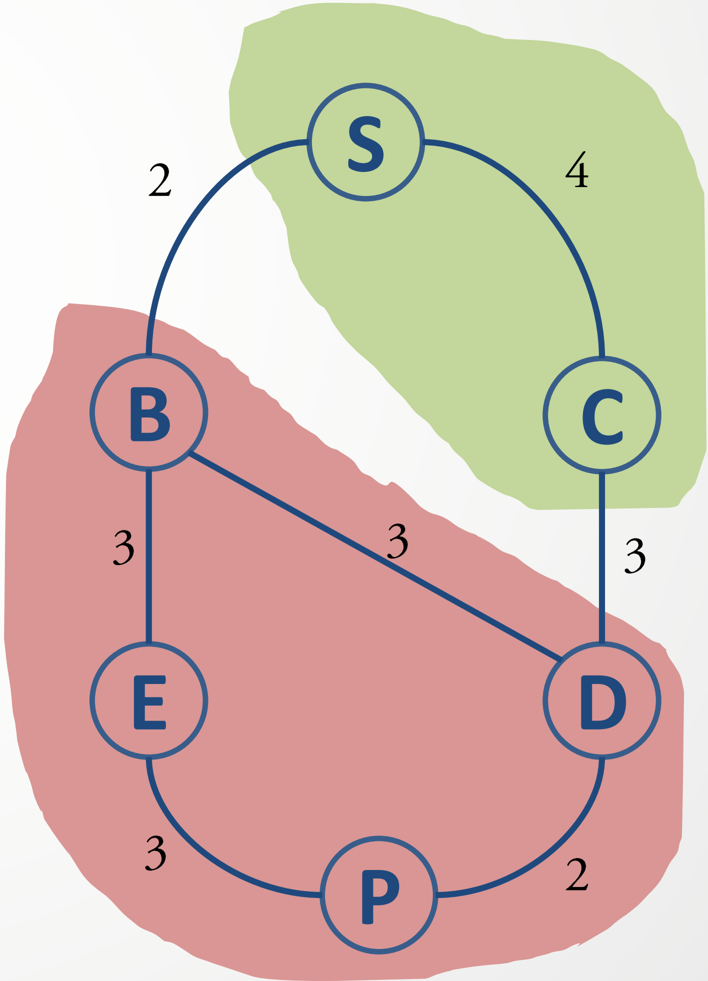
Le problème de la coupe minimale est intimement lié au **problème du flot maximal** dans un graphe…

On regarde chaque arc dont la valeur de capacité restante (le chiffre en noir) est égal à 0…

et on les supprime.

On regarde ensuite quels sommets sont reliés à la source par un chemin (attention, pas le droit d'emprunter les arcs en sens inverse)…

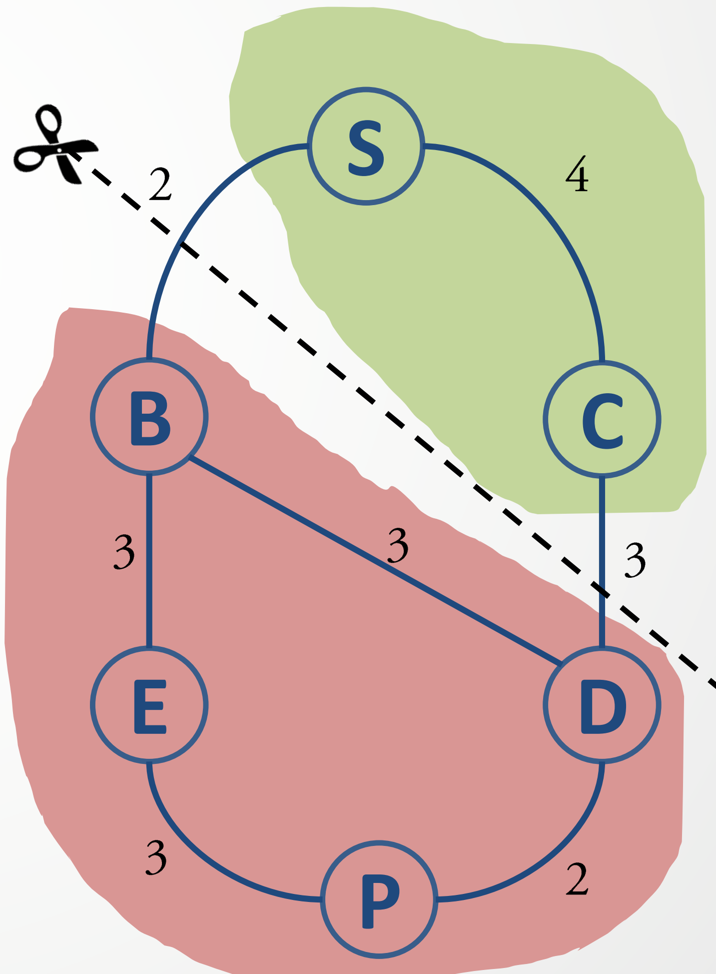
Ici, seul le sommet C est relié à la source.
Ces sommets seront dans le groupe associé à la source, les autre seront dans le groupe du puits.



Coupe minimale et flot

Le problème de la coupe minimale est intimement lié au **problème du flot maximal** dans un graphe…

Ligne de coupe minimale
Poids : 5

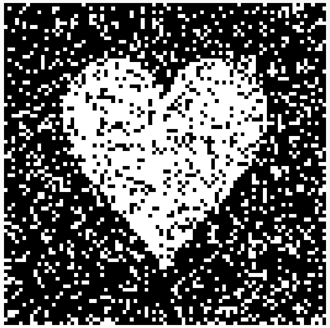


Ce partage permet de trouver une ligne de **coupe minimale.**

Application aux images

Petit rappel

Pour retirer le bruit d'une image binaire,



on souhaite minimiser cette expression en modifiant les valeurs de certains pixels :

$$Score(Im, Ref) = \sum_{p \in Im} |Im(p) - Ref(p)| + \sum_{p,r\ voisins} |Im(p) - Im(r)|$$

Tester toutes les combinaisons possibles de pixels est impossible…

Minimiser le score à l'aide d'une coupe minimale

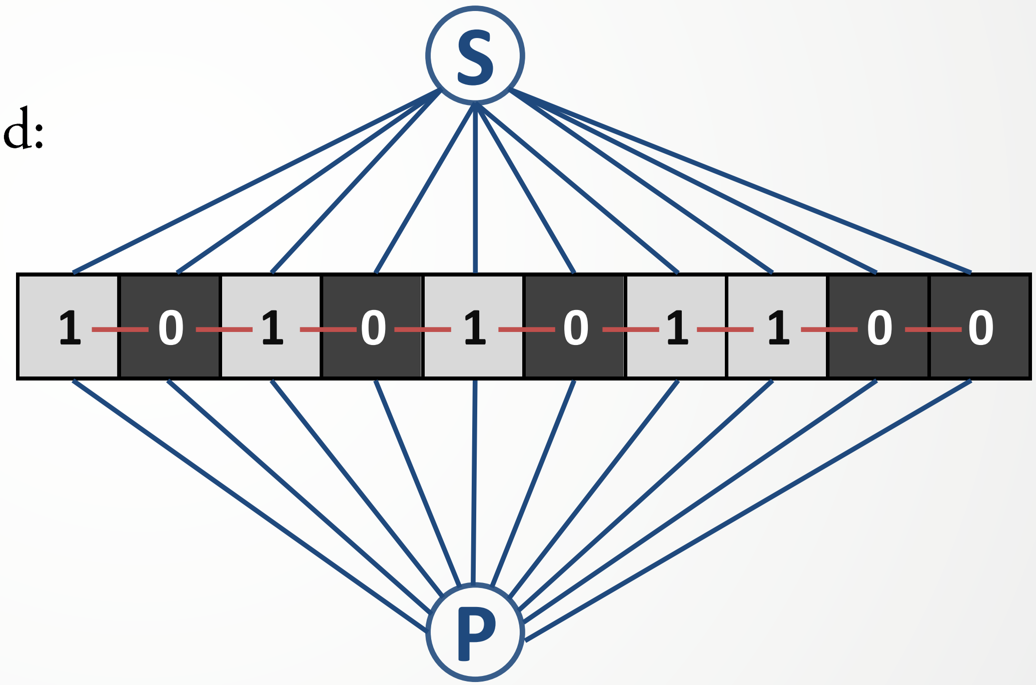
Nous pouvons trouver le score minimum de façon rapide, en utilisant le principe de la coupe minimale…

Prenons par exemple une image 1d:

Chaque pixel sera un sommet du graphe.

Nous connectons une **source** à tous les pixels, et nous connectons tous les pixels à un **puits** (à l'aide d'arêtes)…

Nous **connectons tous les pixels voisins** par une arête…

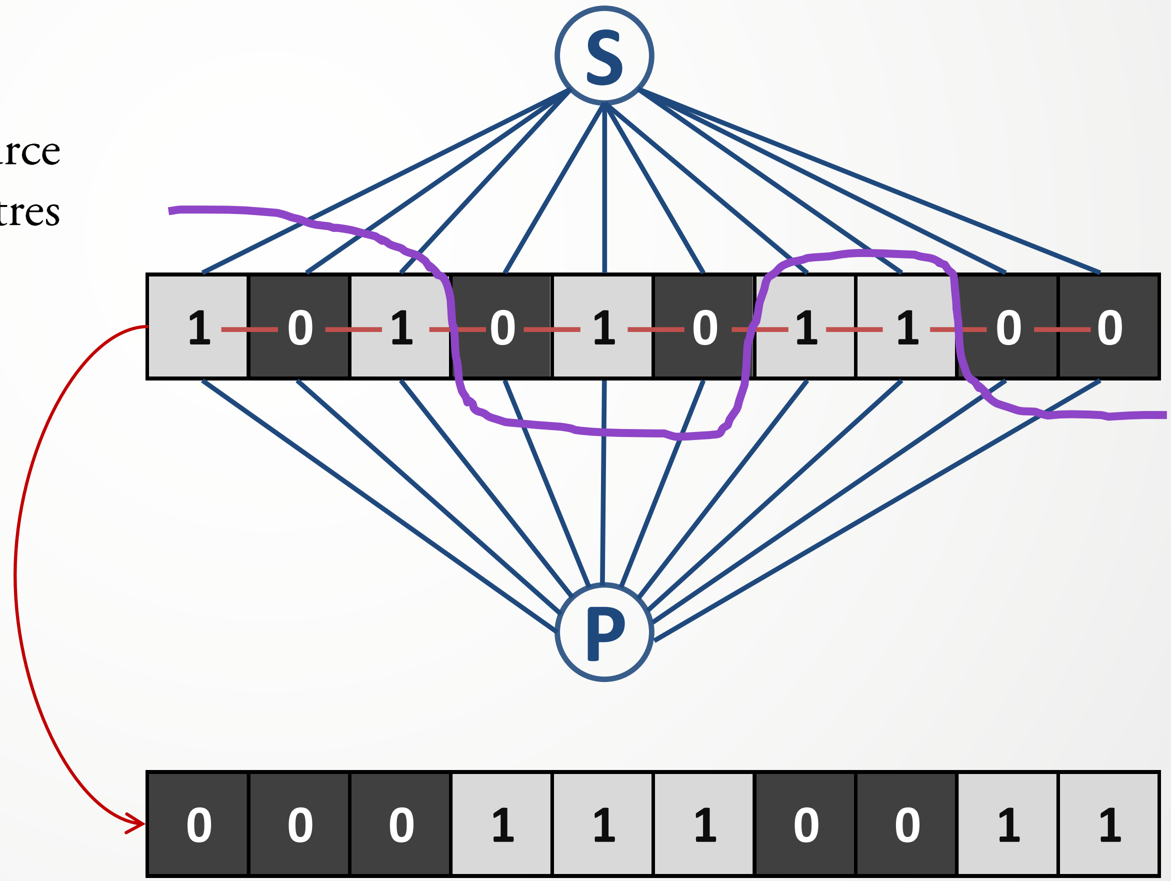


Minimiser le score à l'aide d'une coupe minimale

Nous souhaitons qu'une coupe décide de la couleur à attribuer aux pixels :

Traçons une coupe au hasard :

Chaque pixel connecté à la source deviendra blanc, et les autres deviendront noirs…



Minimiser le score à l'aide d'une coupe minimale

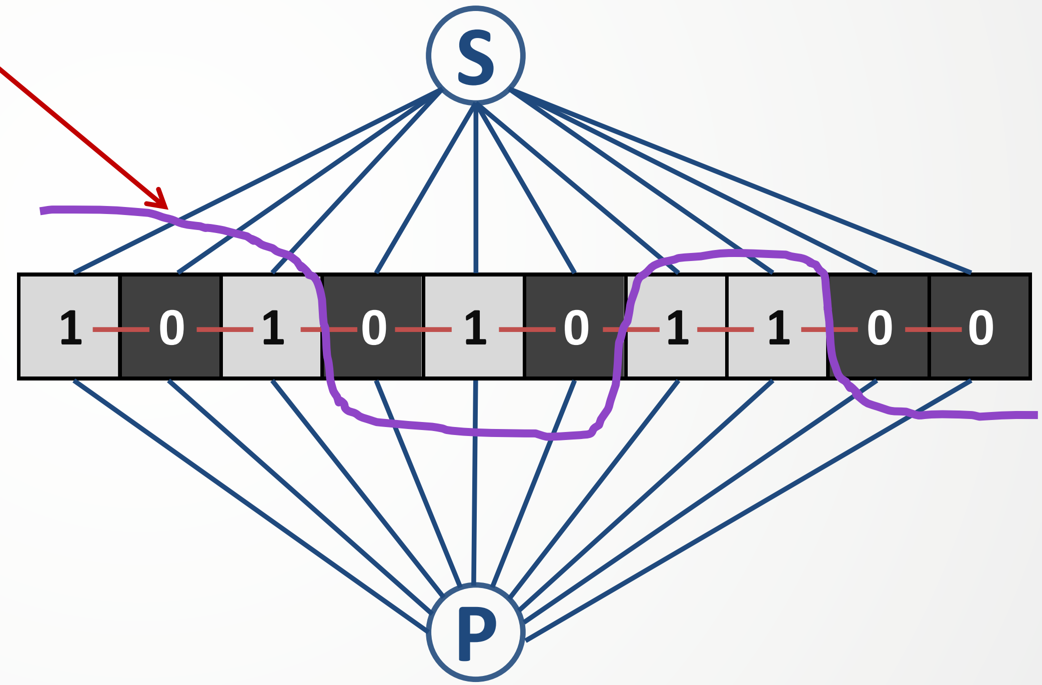
Nous souhaitons qu'une coupe décide de la couleur à attribuer aux pixels :

Si la coupe traverse cette arête, alors elle attribue au pixel correspondant la couleur noire…

…il faut donc qu'elle « paye » le prix pour passer ce pixel en noir.

Le poids de chaque arête reliée à la source sera le coût pour passer le pixel correspondant à 0 (c'est-à-dire 2 pour les pixels valant 1, et 0 pour les pixels valant 0).

Dans le problème de flot, un poids nul équivaut à effacer l'arête.



0	0	0	1	1	1	0	0	1	1

Minimiser le score à l'aide d'une coupe minimale

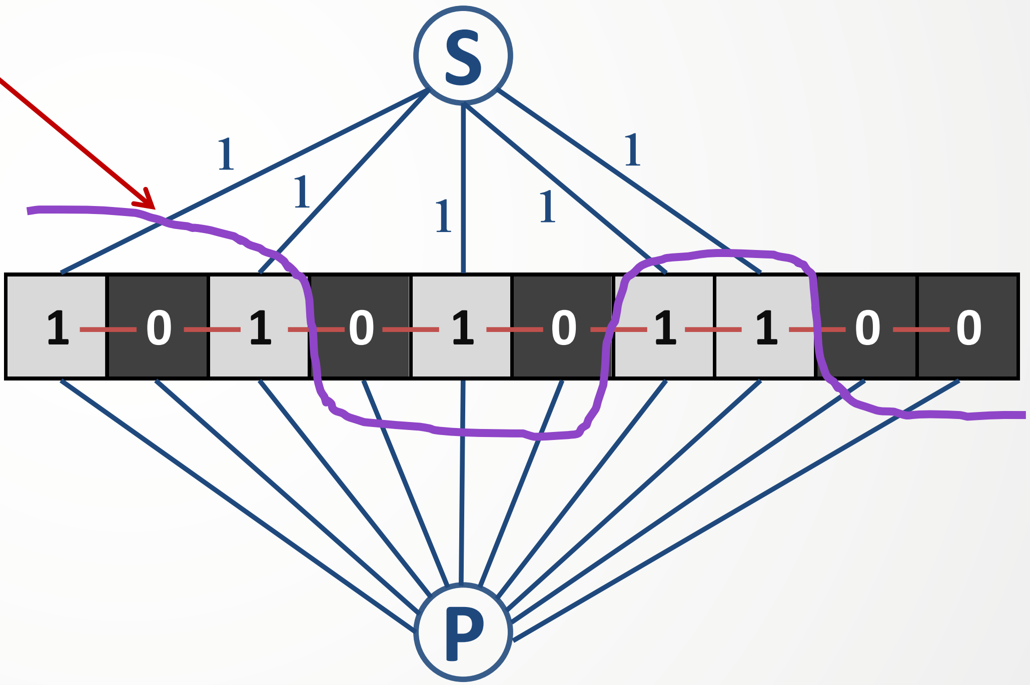
Nous souhaitons qu'une coupe décide de la couleur à attribuer aux pixels :

Si la coupe traverse cette arête, alors elle attribue au pixel correspondant la couleur noire…

…il faut donc qu'elle « paye » le prix pour passer ce pixel en noir.

Le poids de chaque arête reliée à la source sera le coût pour passer le pixel correspondant à 0 (c'est-à-dire 1 pour les pixels valant 1, et 0 pour les pixels valant 0).

Dans le problème de flot, un poids nul équivaut à effacer l'arête.

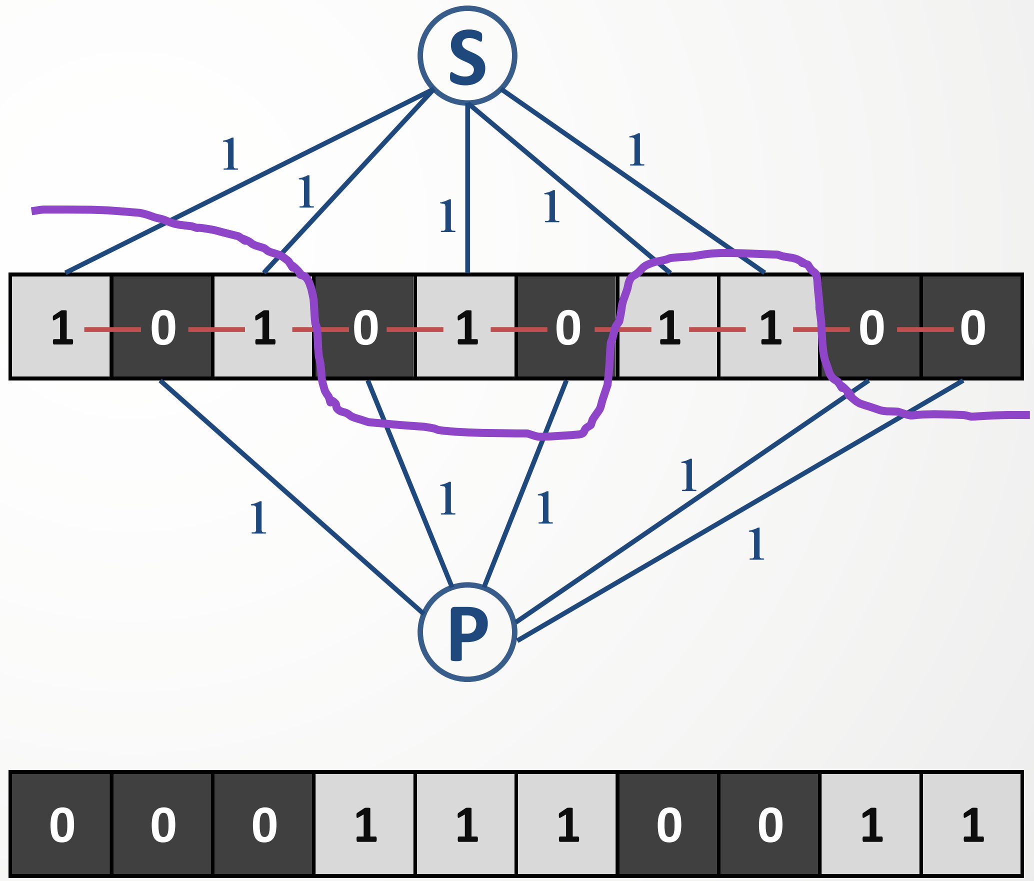


0	0	0	1	1	1	0	0	1	1

Minimiser le score à l'aide d'une coupe minimale

Nous souhaitons qu'une coupe décide de la couleur à attribuer aux pixels :

On fait de même (mais inversement) pour les arêtes allant vers le puits…



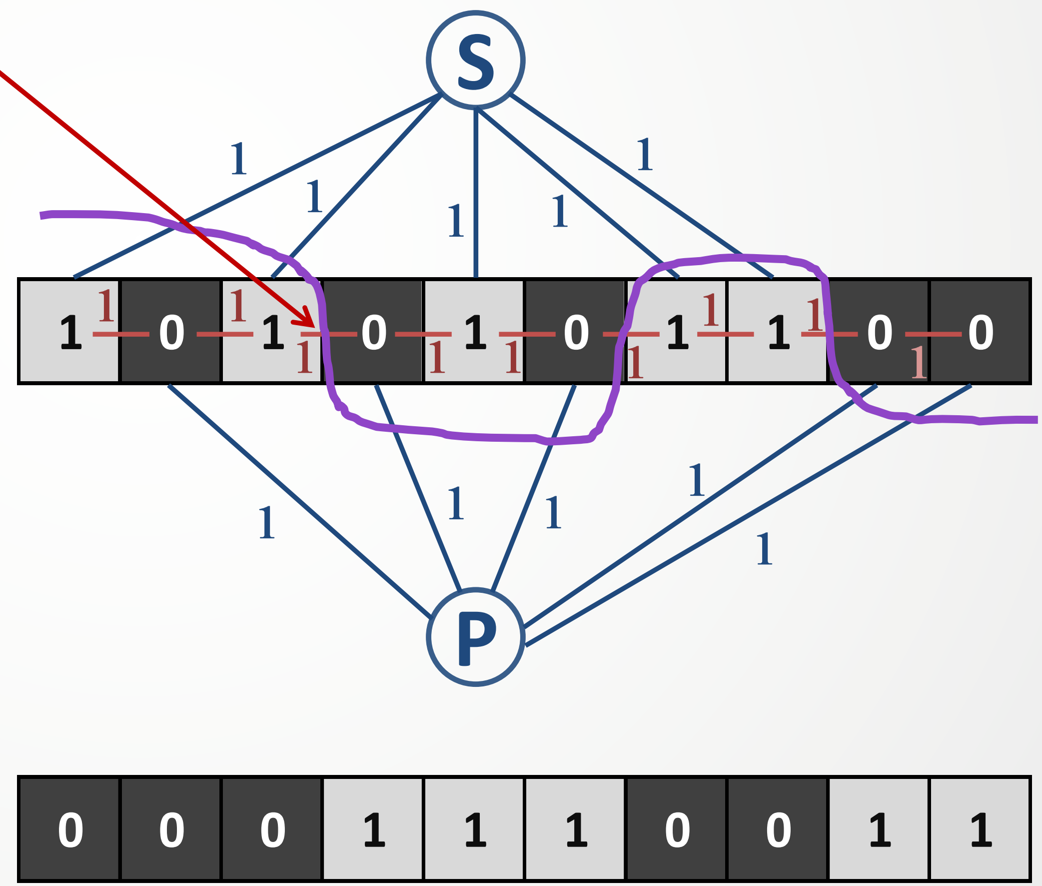
Minimiser le score à l'aide d'une coupe minimale

Nous souhaitons qu'une coupe décide de la couleur à attribuer aux pixels :

Si la coupe traverse cette arête, alors est en train de passer d'un groupe de pixel d'une certaine couleur à un groupe de pixel de l'autre couleur…

…elle doit donc « payer » le prix pour créer des pixels voisins de différentes couleurs.

Le poids de chaque arête horizontale sera le prix à payer pour avoir des voisins de couleurs différentes, c'est-à-dire 1.



Minimiser le score à l'aide d'une coupe minimale

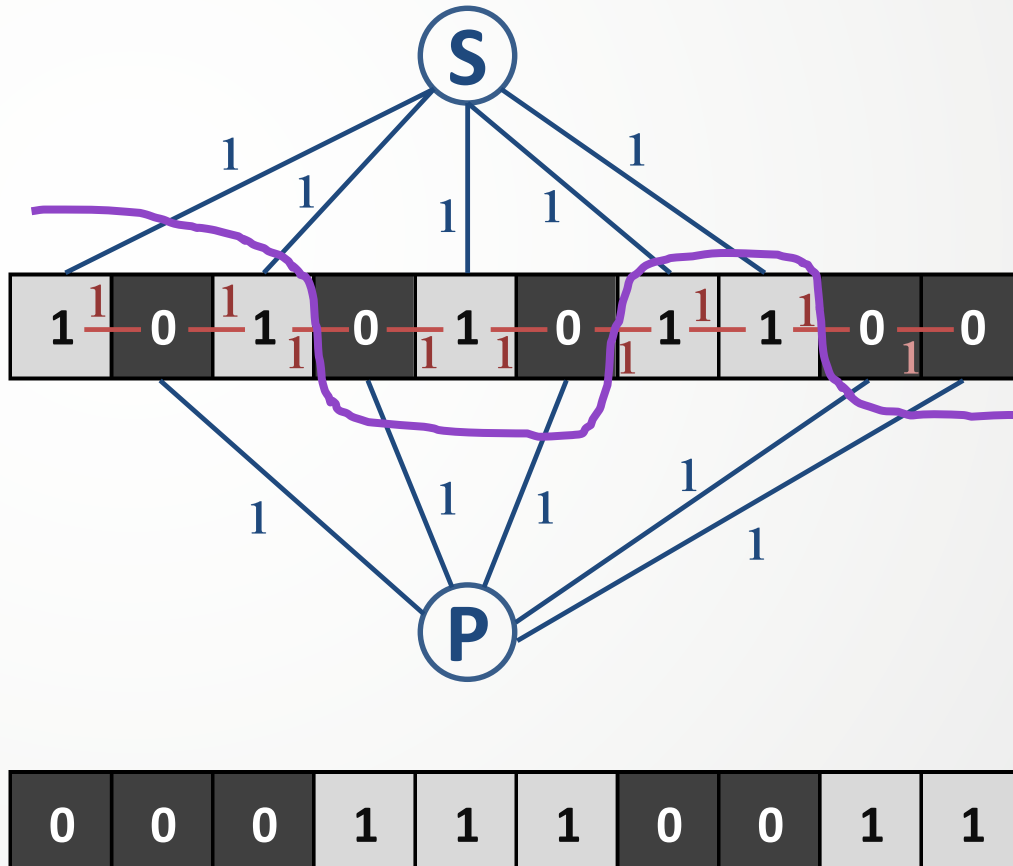
Nous souhaitons qu'une coupe décide de la couleur à attribuer aux pixels :

La coupe minimale sera celle qui regroupera les pixels en deux camps (blanc ou noir) en minimisant le coût des arêtes traversées…

…c'est-à-dire en **minimisant le score**.

La coupe minimale produira donc l'image qui minimise le score que l'on avait construit.

Pour obtenir cette coupe minimale, on fera un algorithme de flot.

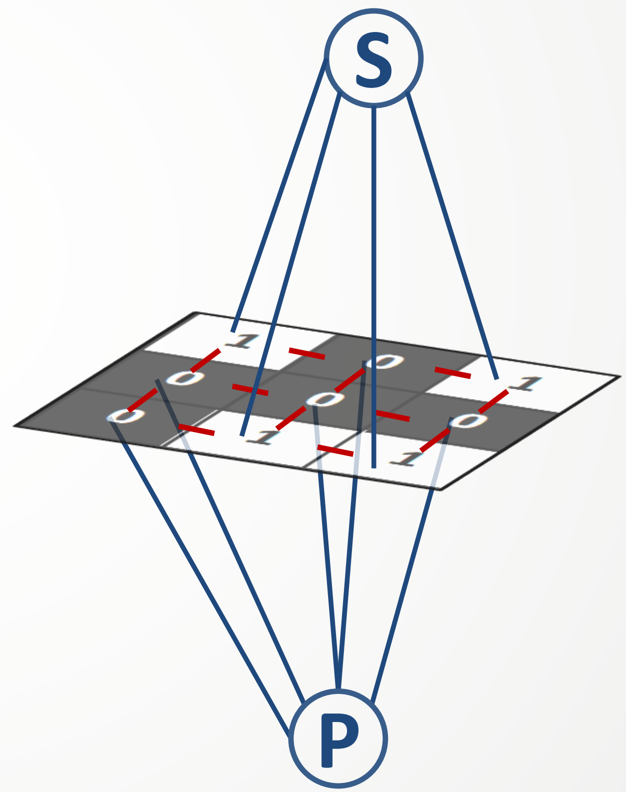


Le cas des images 2d

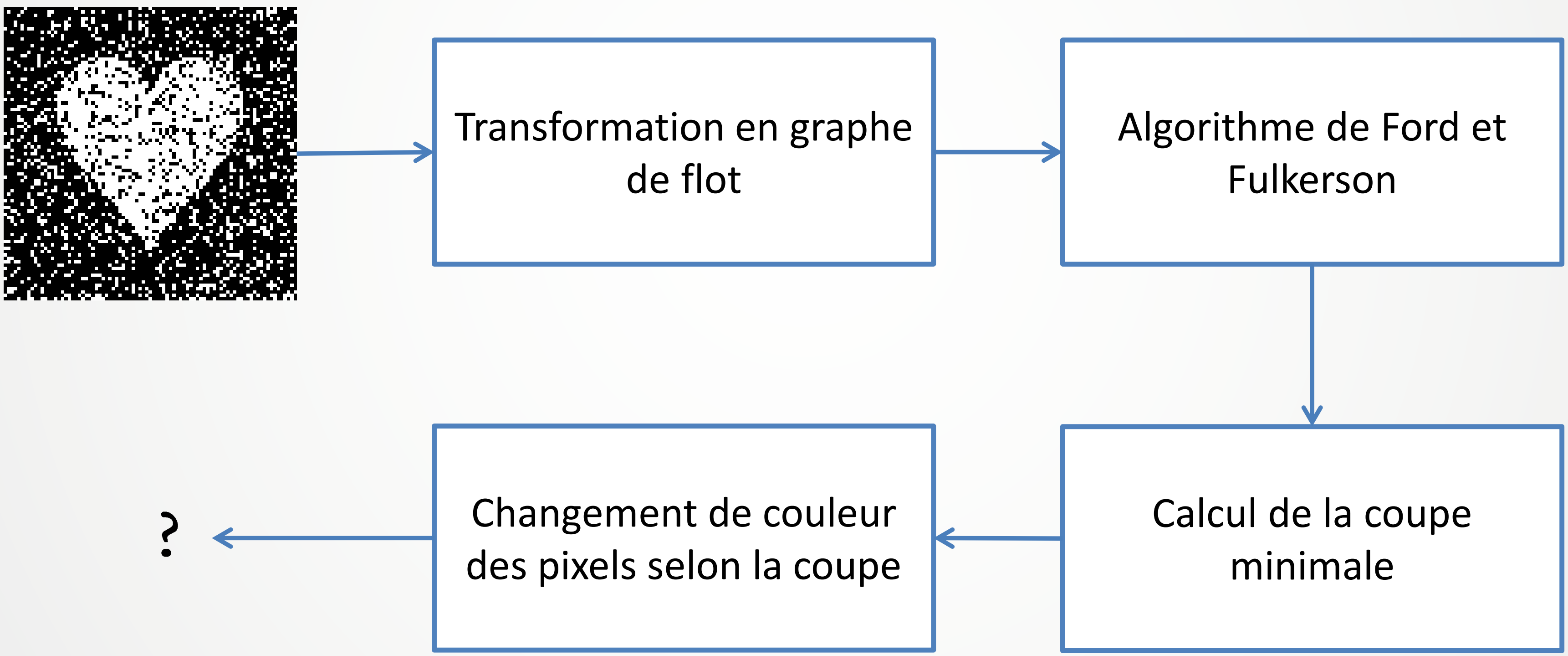
Pour une image 2d, la construction est similaire…

On relie une source et un puits à tous les pixels par des arêtes de poids 1.

On relie chaque pixel voisin par des arêtes de poids 1.

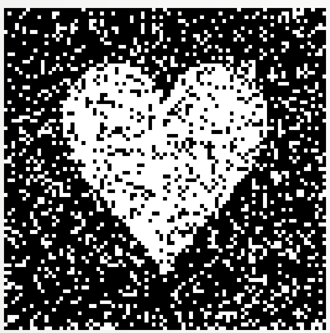


Méthode

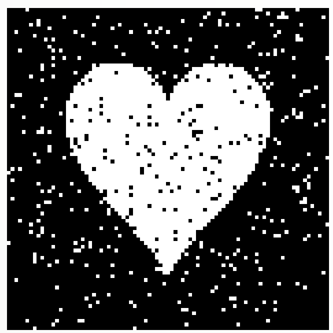

Transformation en graphe de flot
Algorithme de Ford et Fulkerson
Calcul de la coupe minimale
Changement de couleur des pixels selon la coupe
?

Résultat

Résultat trouvé par la coupe minimale



Score : 4782



Score : 3454

Score : 1717

Score : 1690

Modifier le score

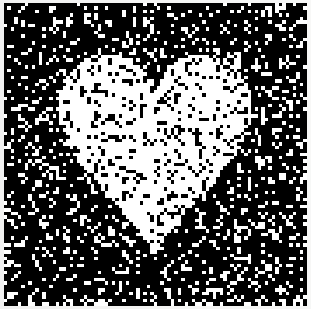
On peut modifier le score de façon à introduire un paramètre lambda qui fera varier l'importance du terme de régularisation par rapport au terme d'attache :

$$Score(Im, Ref) = \sum_{p \in Im} |Im(p) - Ref(p)| + \lambda \sum_{p,r\ voisins} |Im(p) - Im(r)|$$

En faisant varier ce terme, on fait varier la contribution des « pixels voisins de différentes couleurs » par rapport à la contribution des « pixels qui changent par rapport à l'image originale ».

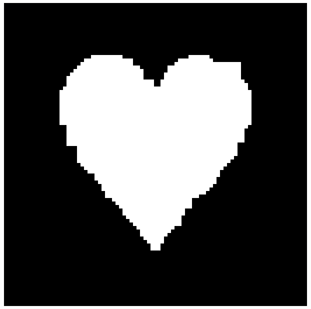
Résultat

Résultats trouvés pour différentes valeur de lambda

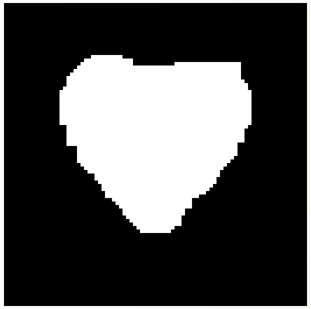


$\lambda = 0$

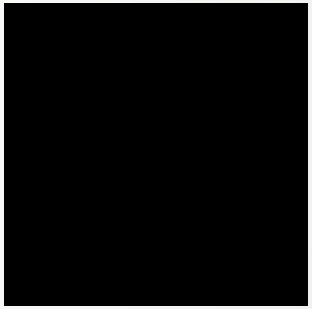
$\lambda = 0{,}4$



$\lambda = 0{,}6$



$\lambda = 3$



$\lambda = 6$

13

Conclusion

Cas des images couleur

Dans le cas des images couleurs (ou même en niveaux de gris), il faut définir une métrique différente pour mesurer les distances entre pixels et source.

Une solution consiste à demander à l'utilisateur de placer des marqueurs (sous forme de boîte englobantes) autour de l'objet d'intérêt et dans l'objet d'intérêt.

Ensuite, à l'aide d'un mélange de gaussienne (avec un certain nombre de gaussiennes à fixer), on estime la distribution des couleurs sur l'image. On utilise ces gaussiennes pour calculer la probabilité d'un pixel d'appartenir au fond ou à l'objet.

Cas de plusieurs objets

Dans le cas où l'on souhaiterait gérer plusieurs marqueurs, une technique utilisée est le « one vs all ».

Pour chaque source, on réalise l'algorithme de graph cut en plaçant la source « contre » toutes les autres sources.

A chaque étape, on calcule l'énergie totale obtenue : si on a réussi à la diminuer, on conserve le nouveau résultat. Sinon, on le rejette.

Quand plus rien ne bouge (ou plus trop), on s'arrête.

Conclusion

Les graph cuts fonctionnent grâce à des **algorithmes simples à comprendre** sur les graphes.

Les résultats de ces algorithmes apportent une **solution au problème de la coupe minimale**, facile à comprendre.

En **adaptant un problème donné** (débruiter une image) en un problème de coupe minimale, on peut le résoudre facilement avec un algorithme de flot.

Beaucoup **d'autres applications** des graph cuts…

…et beaucoup d'autres algorithmes intéressants dans le domaine de l'image.