UNIVERSITÉ DE STRASBOURG

Théorie des Graphes
Fermetures, Connexité, Coupes

Fabrice Theoleyre

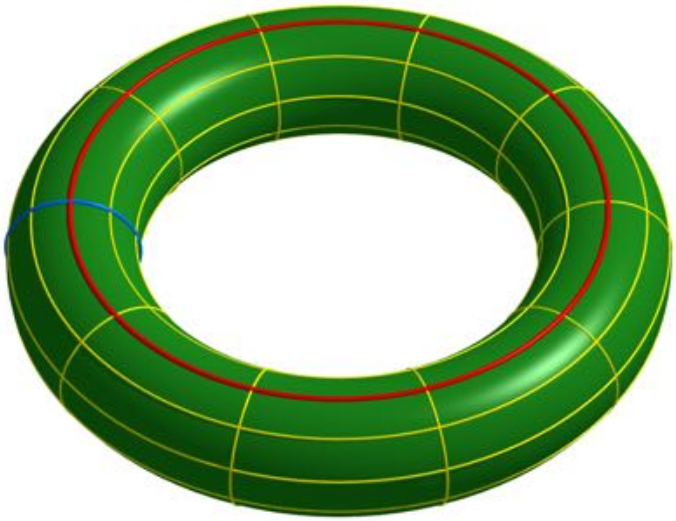
theoleyre@unistra.fr
http://www.theoleyre.eu

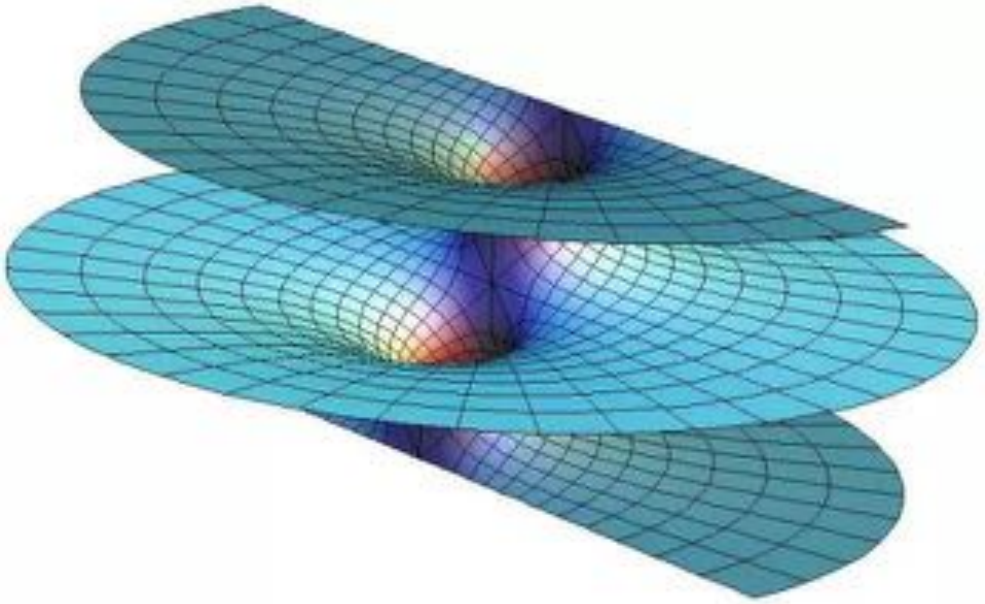
VERS LA CONNEXITÉ

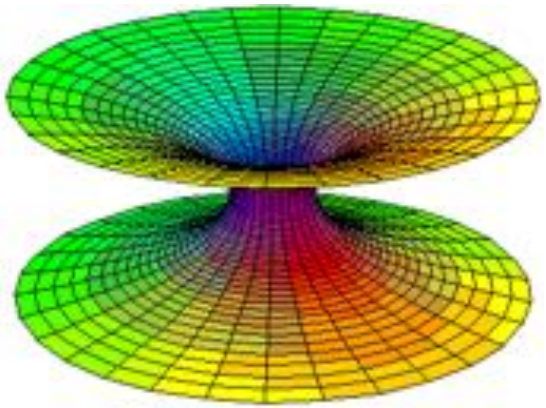
Distance, chemin & chaine

Notion de connexité

- Topologie (maths)
  - ❖ d'un seul tenant





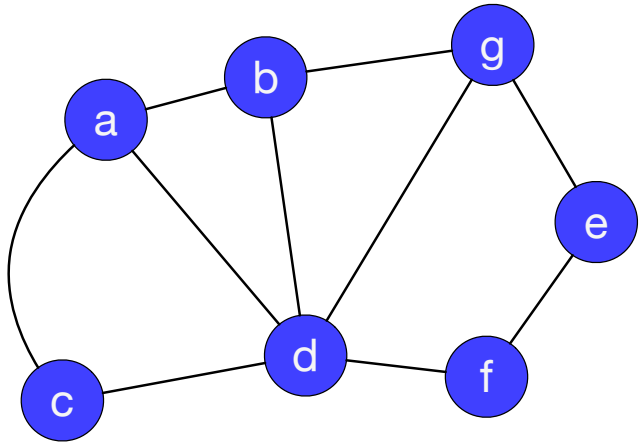


- Graphe (non) orienté
  - ❖ Je peux *connecter* deux sommets en restant dans mon graphe

Chaines dans les graphes non orientés (1)

- Arêtes consécutives
  - (s,t) et (s',t') sont consécutives ssi t=s'

- G=(S,A) un graphe non orienté
  - u,v ∈ S deux sommets
  - une chaine de u à v = une suite finie $(c_i)_{i=0}^{t}$ telle que
    - $t \geq 0,\ c_0 = u,\ c_t$ =v
    - $\forall i \in [0, t-1], (c_{i,}, c_{i+1}) \in A$



- Définitions
  - Circuit = chaine de u vers lui même
  - Boucle = circuit d'un seul arc
  - Chaine simple (adbgdc)
    - $\forall\, i,j \in [0,t-1]\ i \neq j, (c_i, c_{i+1}), (c_j, c_{j+1}) \in c, c_i = c_j \Rightarrow c_{i+1} \neq c_{j+1}$
    - N'utilisant que des arêtes différentes
  - Chaine élémentaire (abd)
    - $\forall\, i,j \in [0,t-1]\ i \neq j, (c_i, c_{i+1}) \in c \wedge (c_j, c_{j+1}) \in c,\ c_i \neq c_j$
    - N'utilisant que des sommets différents

Chaines dans les graphes non orientés (2)

- Propriétés
  - S'il existe une chaine de x à y alors
    - ❖ il existe une chaine de y à x (symétrie)
    - ❖ il existe une chaine élémentaire de x à y

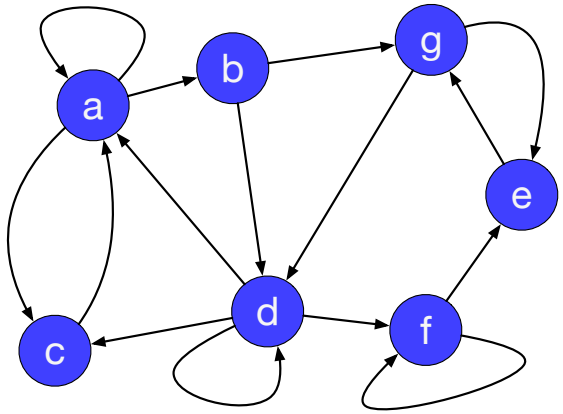
- Distance entre deux sommets
  - Nombre de sommets dans le plus court chemin entre u et v – 1
  - soit $\sigma(u, v)$ l'ensemble des chemins entre u et v
  - $d(u, v) = Min_{c \in \sigma(u,v)}(|c| - 1)$

Chemins dans les graphes orientés

- Chemin = la transposition des chaines aux graphes orientés
  - u,v ∈ S deux sommets
  - Un chemin de u à v = une suite finie ordonnée $(c_i)_{i=0}^{t}$ telle que
    - $t \geq 0$, $c_0 = u$, c_t =v
    - $\forall i \in [0, t-1], (c_{i,}, c_{i+1}) \in A$
  - Le chemin n'est pas obligatoirement symétrique !
    - E.g. mon trajet dans les deux sens ne passe pas forcément par les mêmes arcs

Composantes connexes (1)

- Soit G=(S,A), il s'agit alors des classes d'équivalences de S suivant ~G
  - Plus formellement sur (S, ~G), il s'agit :
    - des ensembles $\forall x \in S, [x]_{\sim_G} = \{y \in S \mid \exists\ chaine\ c(x,y)$ dans G$\}$



  - graphe **faiblement** connexe
    - Il existe une **chaine** entre toute paire de sommets
    - (on ne regarde pas le sens des arcs)
  - Graphe **fortement** connexe
    - Il existe un **chemin** entre toute paire de sommets
    - sa fermeture transitive est complète, elle forme une clique
    - intuitivement, il existe un chemin entre toutes paires de sommets dans le graphe
  - **Composante** connexe
    - Sous-graphe connexe maximal pour l'inclusion

Composantes connexes (2)

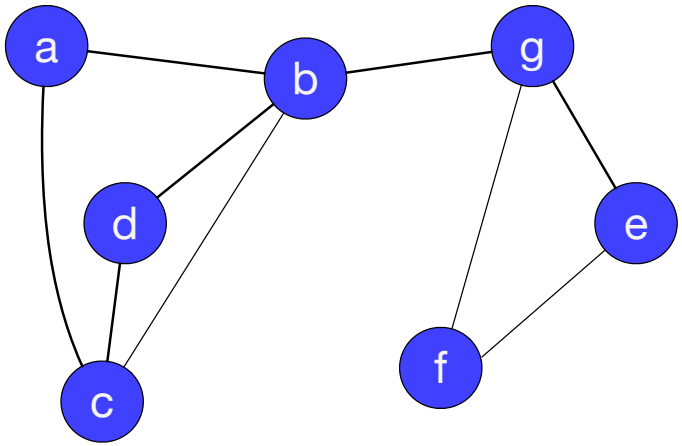
- Graphe biconnexe ?
  - Il existe toujours deux chemins / chaines
  - Extension à la k-connexité

- Découpage en composantes connexes
  - Notion d'isthme
    - ❖ Si on enlève cette arête → plus de connexité
  - Point d'articulation
    - ❖ Sommet dont le retrait augmente le nombre de composantes connexes
    - ❖ Notion de coupe (cf. juste après)

- Calcul des composantes fortement connexes
  - algorithme de Tarjan (linéaire en |A| ou |S|)
  - algorithme de Kosaraju : plus simple à prouver que Tarjan mais moins efficace
  - Idées d'algorithmes efficaces ?

Détection d'isthmes (ou ponts)

- Comment trouver les isthmes dans un graphe ?
  - Suppression d'une arête → augmentation du nb. de composantes connexes



- Approche naïve
  1. Suppression d'une arête
  2. Le graphe est-il connexe ? (BFS)
  3. Remets l'arête
  4. Complexité : O(E * (V + E)) avec une liste d'adjacence

- Implémentation plus efficace ?

FERMETURES

Fermetures

- Fermeture Réflexive
  - Pour un graphe G=(S,A), il s'agit du graphe $G_r=(S,A_r)$ tel que
    - ❖ $A_r = A \cup \{(x,x) \mid x \in S\}$
    - ❖ J'ajoute une boucle sur chaque sommet
  - Algorithme ?
    - ❖ Codage matriciel
      - Union du graphe R et de la diagonale de X

- Fermeture Symétrique
  - J'ajoute une arête inverse
  - Définition formelle ?
    - ❖ $\forall x, y \in X, \forall (xRy) \rightarrow \exists (yRx)$
  - Algorithme ?

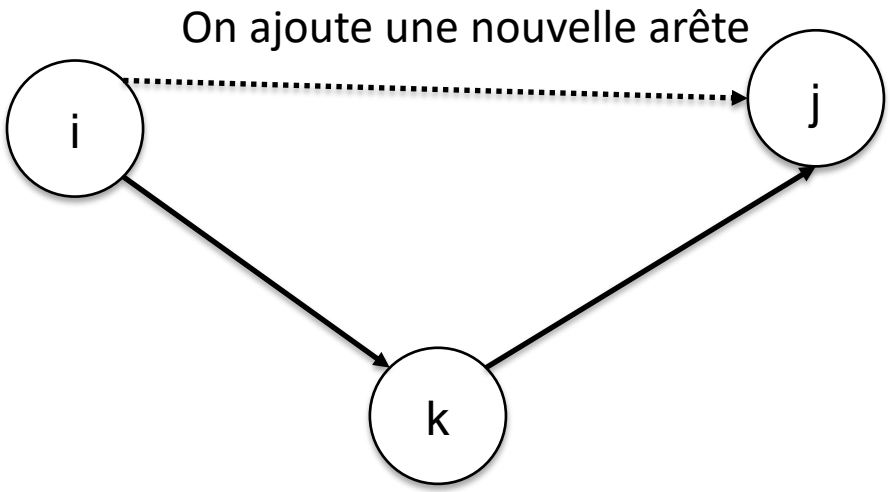
```
Pour u,v in S
  Si C[u,v] ou C[v,u]
     C[v,u] = 1;
     C[u,v] = 1;
  Fsi
Fpour
```

Fermetures (suite)

- Fermeture Transitive
  - la plus petite relation transitive contenant la relation de départ
    - ❖ $R^+ = \bigcup_{k \geq 1} R^k$
  - Définition formelle ?
    - ❖ Définition « graphe »
      - R_1=R
      - $R_{i+1} = R_i \cup \{ (x,z) | \exists y \in S , (x,y) \in R_i, (y,z) \in R \}$
    - ❖ Définition « matricielle »
      - G_0 la matrice identité
      - G_1 la matrice d'adjacence de G
      - $\forall k \in [2..n], G_k[i,j] = G_{k-1}[i,j] \vee (G_{k-1}[i,k] \wedge G_{k-1}[k,j])$
- Notation
  - A^+ : fermeture transitive
  - A^* : fermeture transitive réflexive
    - ❖ $A^* = \bigcup_{k \geq 0} A^k$

Algorithme de Floyd-Warshall

- En réalité algorithme de Roy (antériorité)
- Principe
  - Calcule la fermeture à partie de la matrice d'adjacence C
  - A l'étape k
    - ❖ Calcule un chemin passant par tous les sommets {0..k}
    - ❖ k augmente pour couvrir tous les sommets
  - Programmation dynamique



On parcourt seulement les sommets de 0 à k-1

Algorithme de Floyd-Warshall

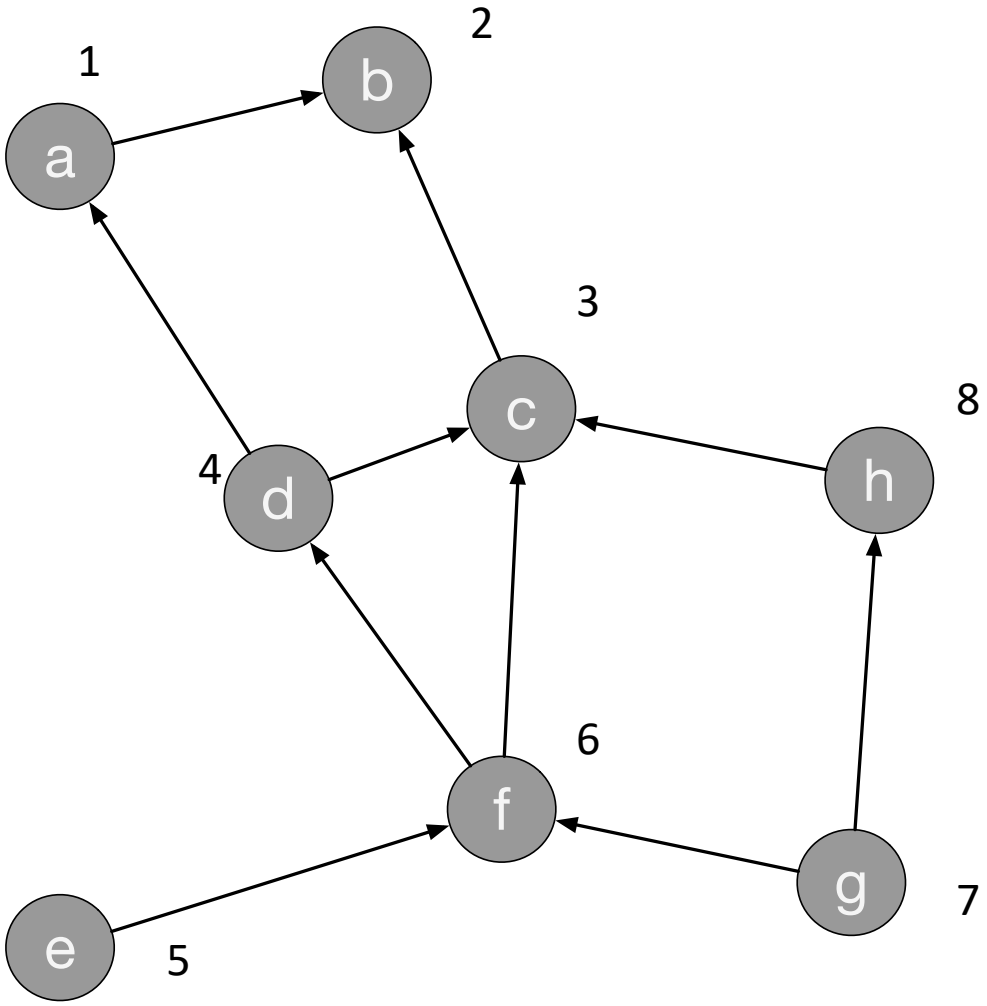
- Principe
  - Calcule une suite de graphes $G_0, \dots, G_n$
  1. $G_0 = G$
  2. G_k a une arête dirigée (i,j) si G possède un chemin de v_i à v_j via des sommets appartenant à l'ensemble $\{v_l\}_{l \in [0,k]}$
  3. $G_n = G^*$ (fermeture transitive)

- Pseudo code
  - pour k de 1 à n
    - ❖ pour i de 1 à n
      - – Si C(i,k)
        - » pour j de 1 à n
          - » Si C(k,j) alors C(i,j) = vrai
    - ❖ retourner C

- Complexité ?
  - O(n^3) en temps, O(n^2) en espace

Algorithme de Floyd-Warshall

- Exemple



Fermeture transitive & graphes

- La relation $\sim_G$ définie sur l'ensemble des sommets S d'un graphe non orienté G=(S,A) par $x \sim_G y$ ssi il existe une chaine de x à y dans G est une relation d'équivalence sur S
  - Preuve
    - ❖ Symétrie
    - ❖ Transitivité
    - ❖ Réflexive

Symétrie et Propriétés

- Le symétrique d'un graphe $G^{-1} = (S, A^{-1})$ est tel que $A^{-1} = \{(x,y) \in S^2 \mid (y,x) \in A\}$
  - ~ $\forall x \in S, R^{-1}(x) = \{y \in S \mid x \in R(y)\}$

- Lien entre graphe et relation
  - Un graphe est symétrique si sa relation A l'est
  - Un graphe est antisymétrique si sa relation A l'est
  - etc.

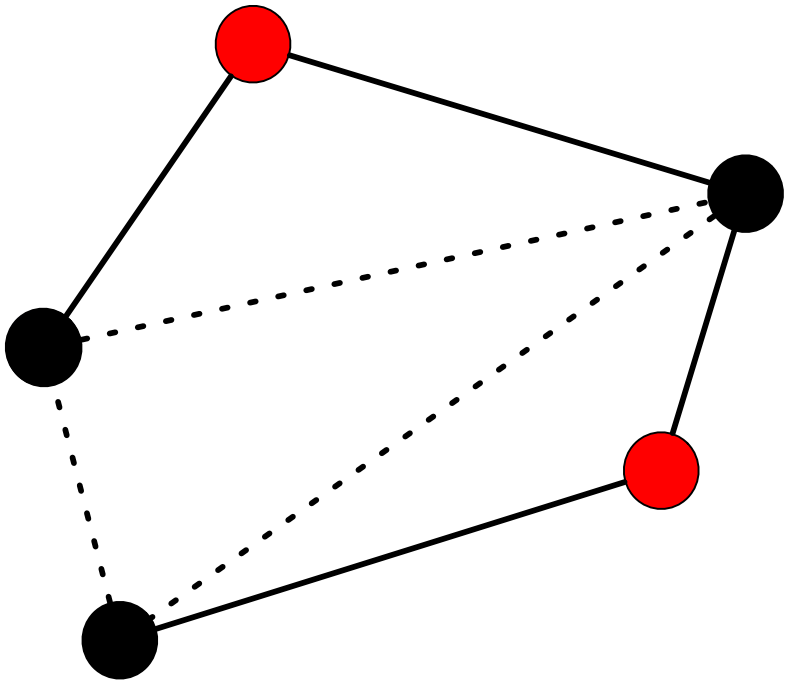
- Identique pour les fermetures
  - notion de symétrisé = (A ou A^{-1}), un graphe non orienté
  - Fermeture Transitive = Graphes complets (cliques) si connexité forte (une seule composante fortement connexe)

QUELQUES OPÉRATEURS

Caractérisant un graphe

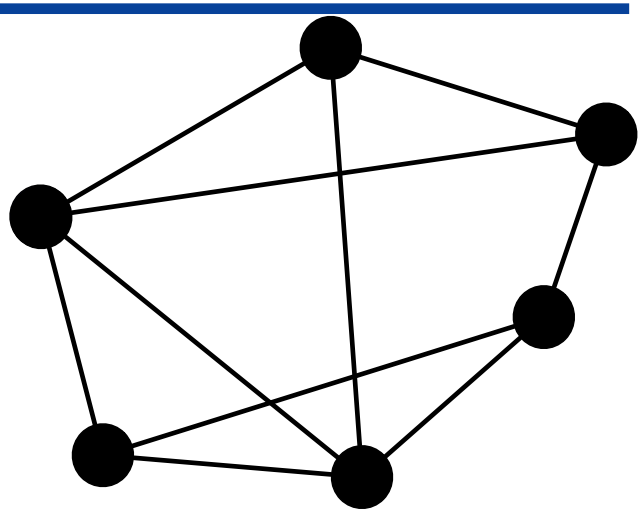
Coupe

- Coupe = créer 2 partitions (disjointes) de G(S,A)
  - Choisir une couleur pour chaque sommet (rouge ou noir)
    - ❖ $S = V_1 \cup V_2$ et $V_1 \cap V_2 = \emptyset$
  - Coupe C
    - ❖ Ensemble des arêtes $C = \{(u,v) | \, u \in V_1 \wedge v \in V_2\}$
  - Poids de la coupe = somme des poids des arêtes de la coupe
  - $Poids = |(u,v) \, | \, u \in V_1, v \in V_2, (u,v) \in A \, |$
- Analogie avec les graphes biparti
  - Le graphe G(V,C) représente un graphe biparti



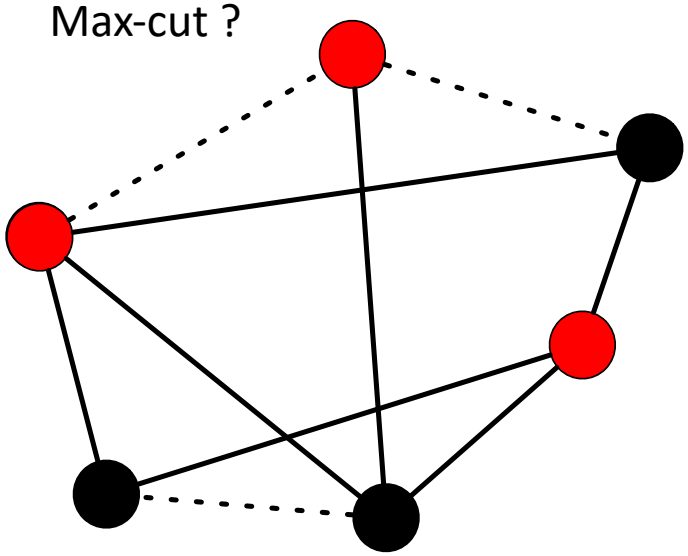
Coupe Maximum

- MAX-CUT
  - Coupe au moins aussi grande que toutes les autres coupes
  - NP-Complet
    - ❖ 21 problèmes de Karp



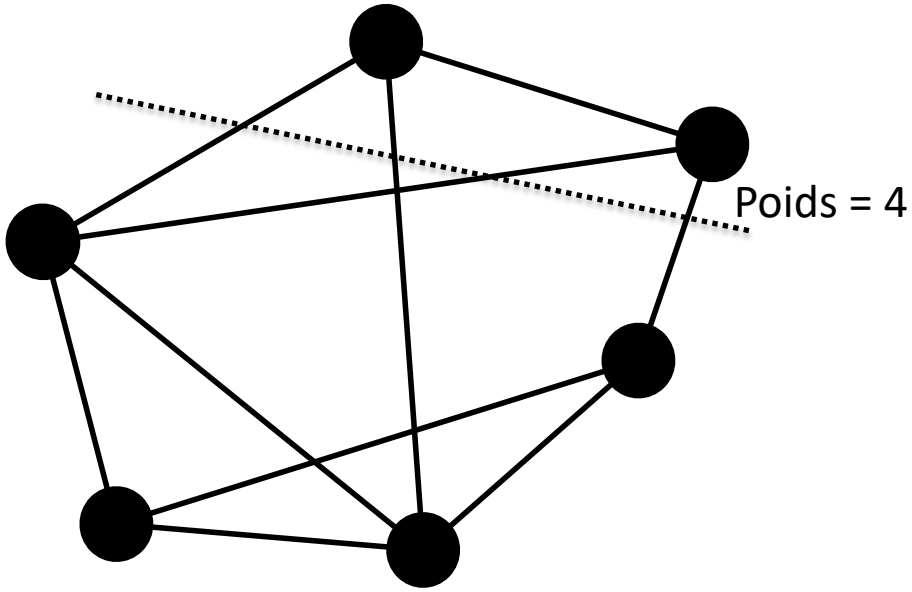
- Extensions
  - Graphe pondéré : somme des poids des arêtes de la coupe
  - k-coupe : partition jusqu'à k composantes

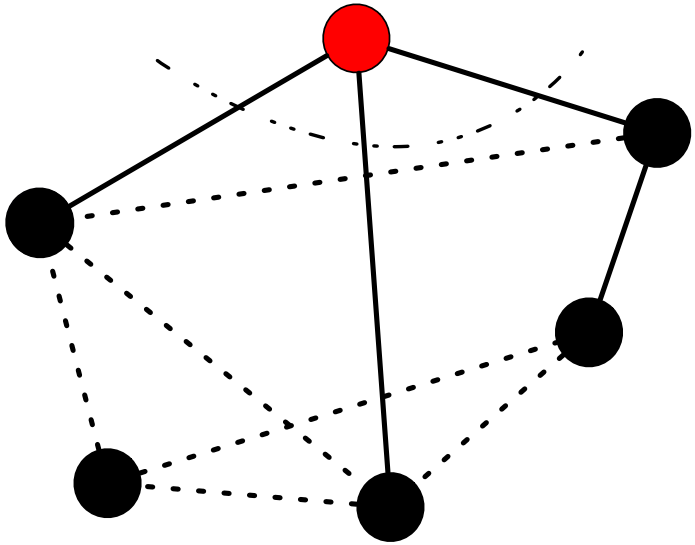
Max-cut ?



Coupe Minimum

- MIN-CUT
  - 1 partition en 2 ensembles minimisant le poids de la coupe
  - Chaque ensemble a une cardinalité strictement positive





EXPLORATION D'UN GRAPHE

Notion de parcours

Exploration d'un graphe

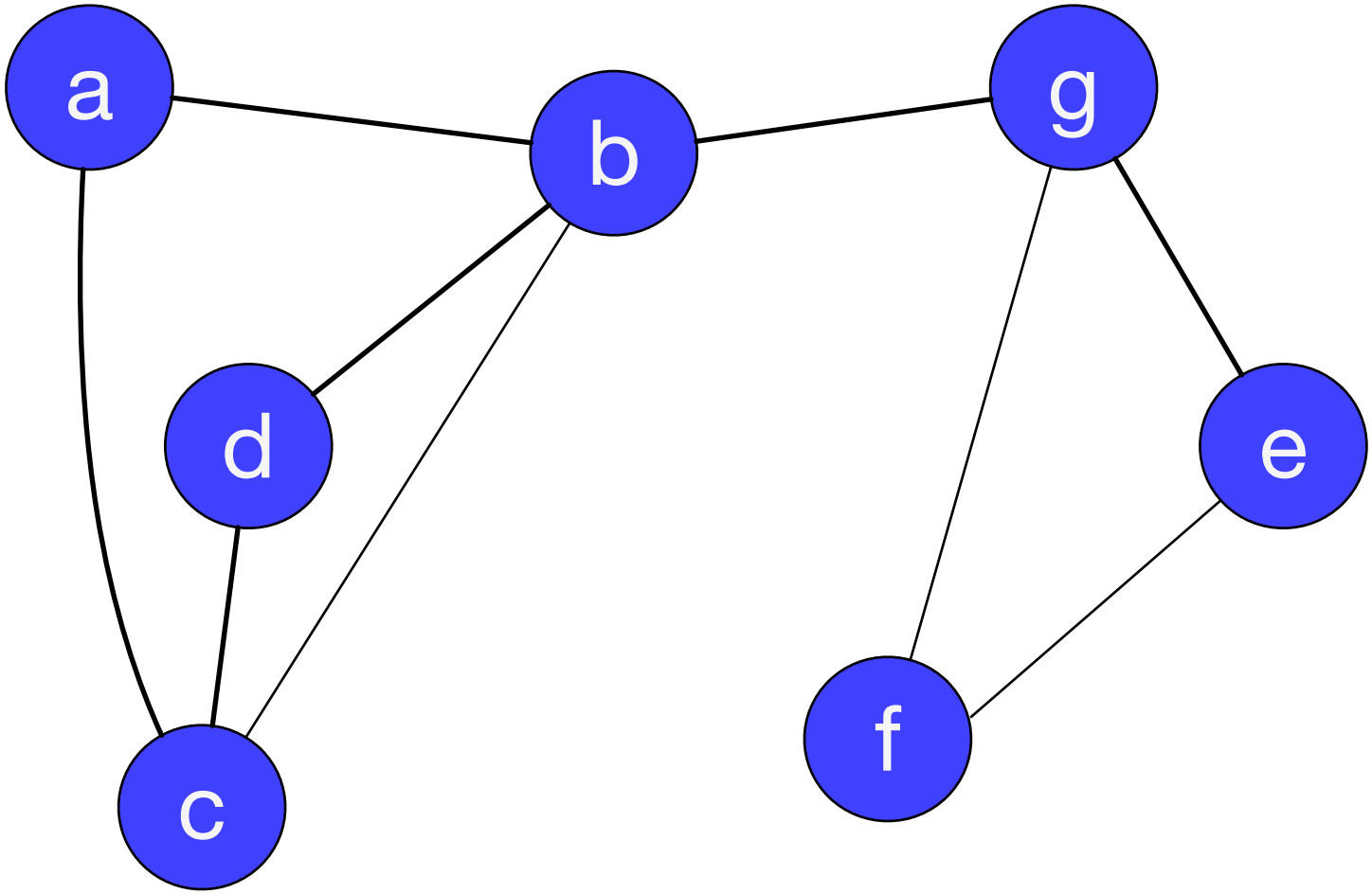
- Passer par tous les sommets
  - Une et une seule fois

- Parcours en profondeur
  - Depth First Search (DFS)
  - Parcours récursif
  - J'explore dans une direction jusqu'à aboutir à une impasse
    - ❖ Plus d'arête
    - ❖ Ou sommet déjà explore

- parcoursProfondeur(G, moi)
  1. Marquer (moi)
  2. Pour tout sommet s non marqué voisin de moi dans G
     1. parcoursProfondeur(G, s)

Exemple de parcours en Profondeur

- En partant de b



- b,a,c,d,g,e,f

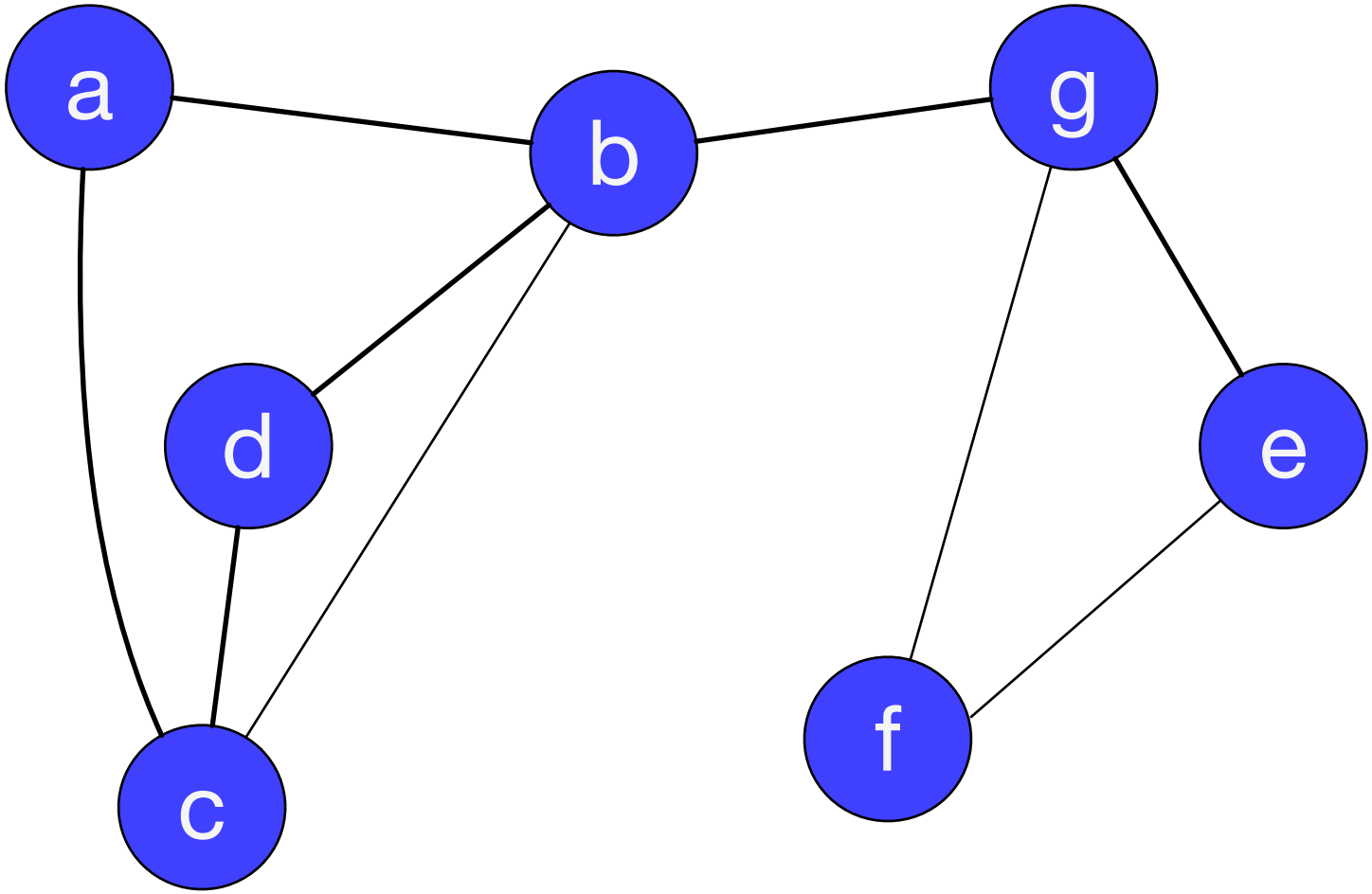
Parcours en Largeur

- Breadth First Search (BFS)
  - j'explore tous mes voisins, puis les voisins de mes voisins, etc.
  - Parcours itératif aisé

- ParcoursLargeur(graph, racine)
  1. InitialiserListeFIFO(l)
  2. marquer(racine)
  3. empilerListe(l, racine)
  4. Tant que l est non vide
     1. Elem = dépilerListe (l)
     2. Pour tout voisin non marqué de elem dans G
        1. EmpilerListe(f, voisin)
        2. Marquer(voisin)

Exemple de parcours en Largeur

- En partant de b



- b,a,d,g,c,f,e