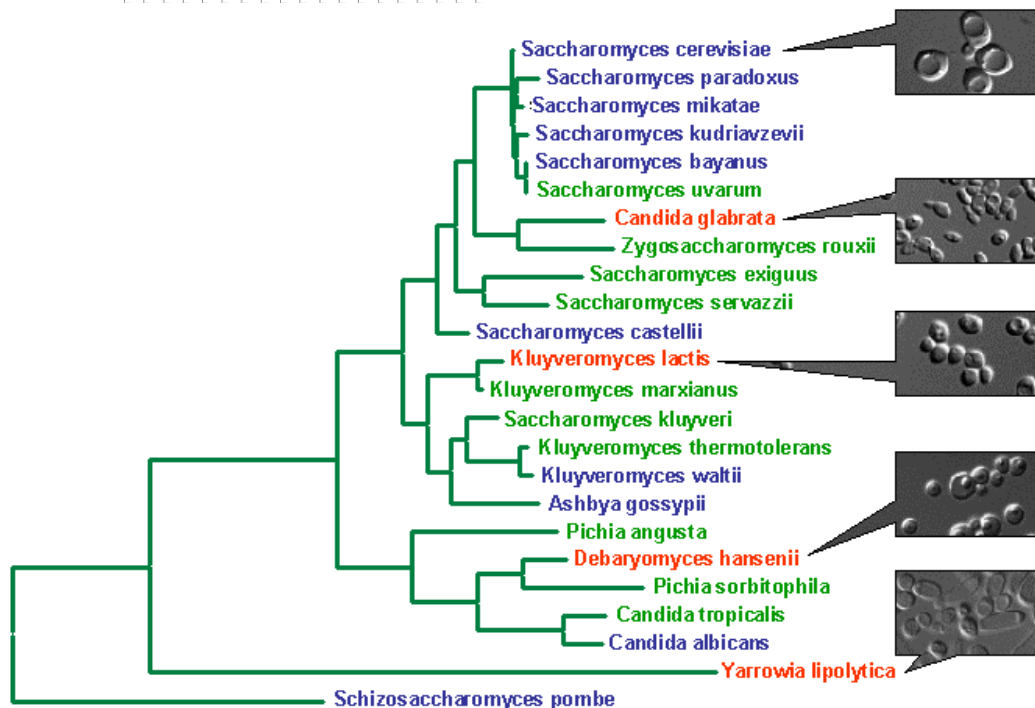
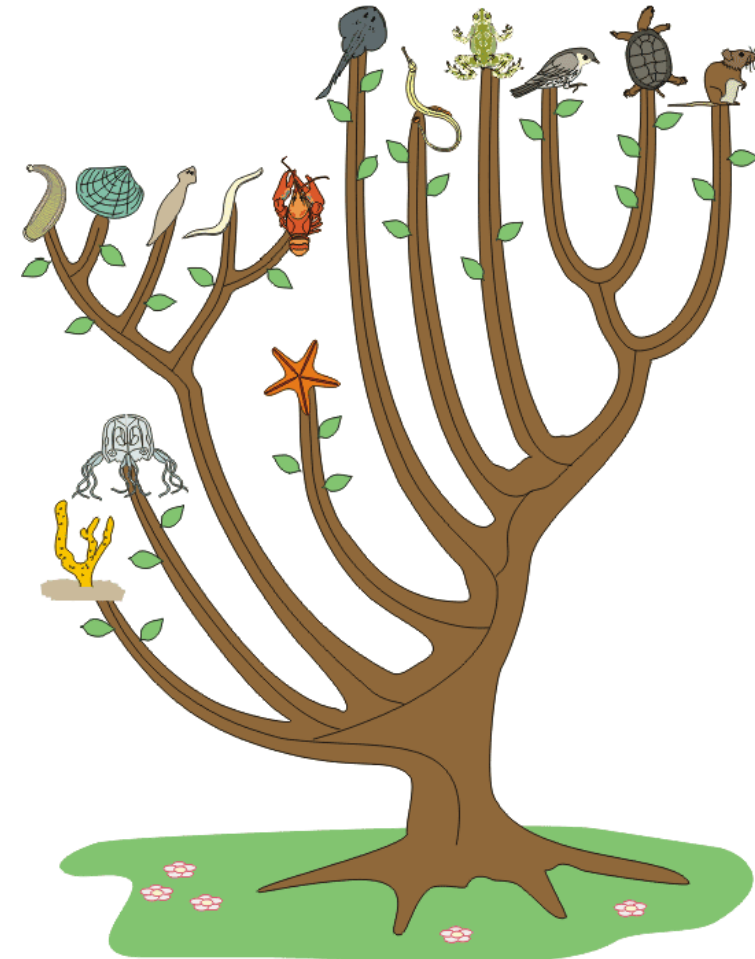
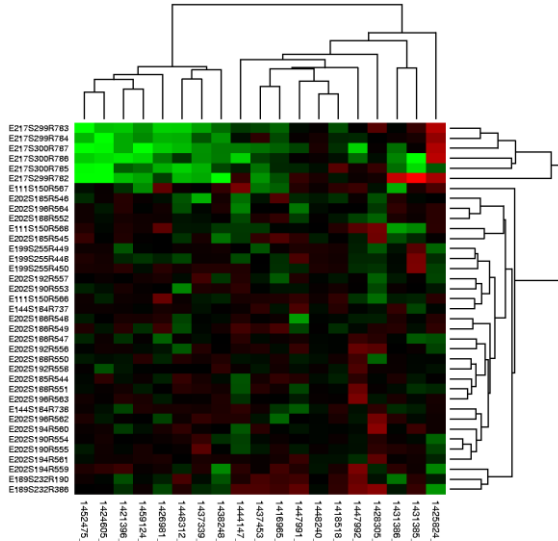


Traitement des graphes et réseaux biologiques

Master 1 MABS



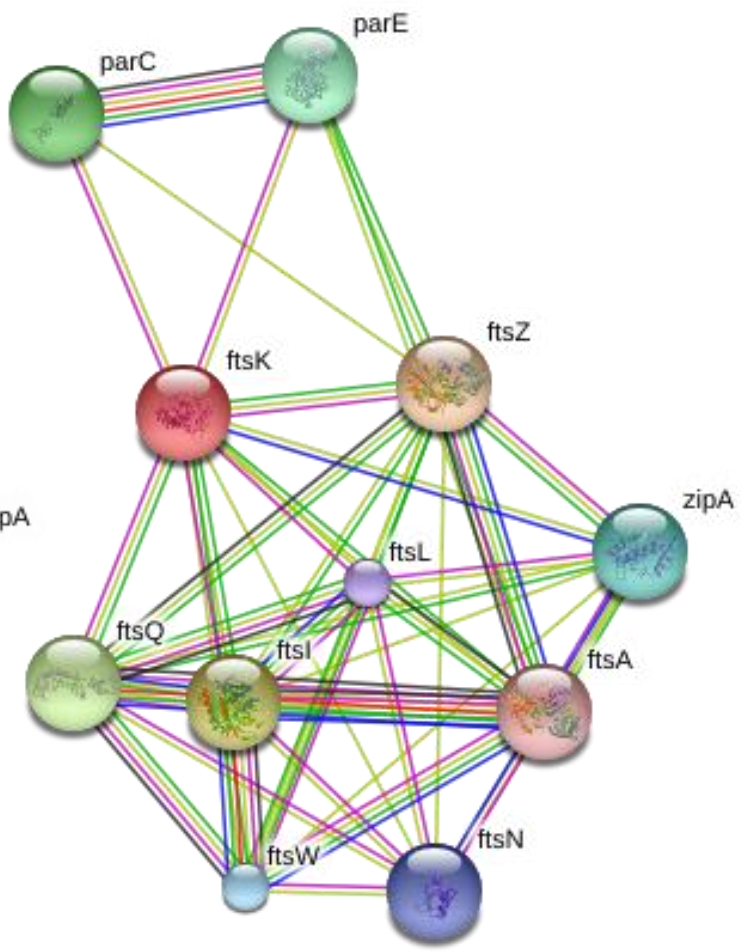
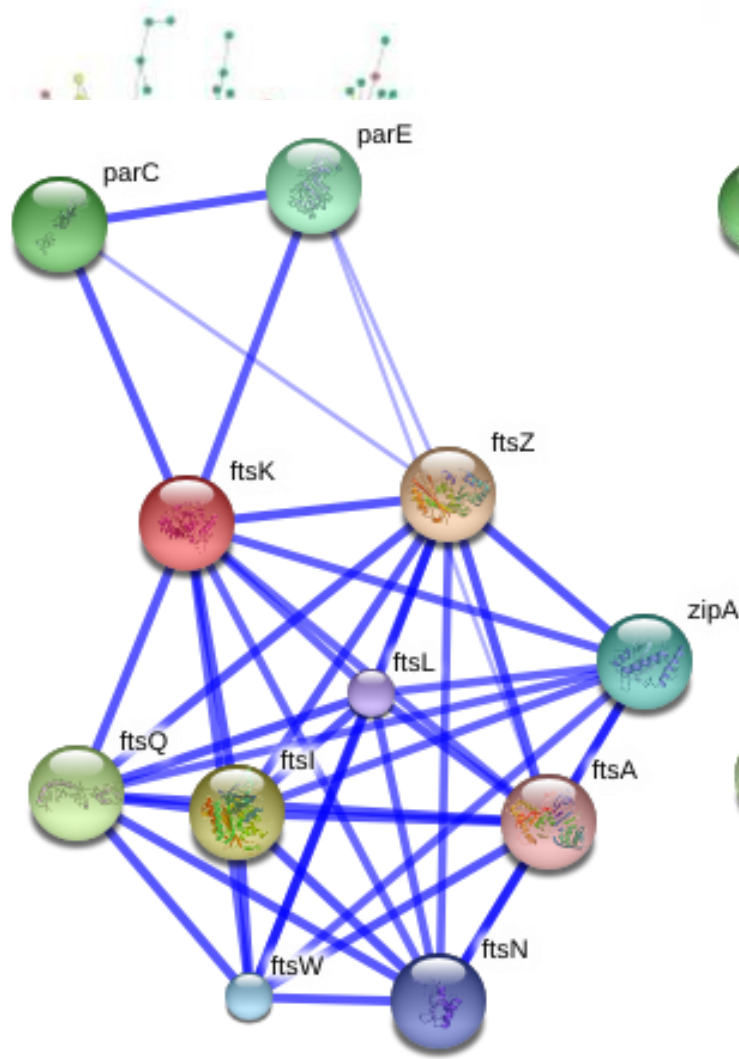
Graphes et réseaux biologiques



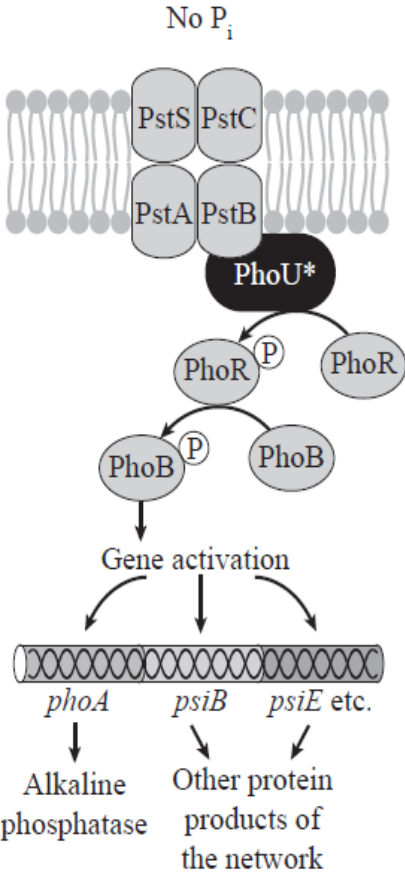
Graphes et réseaux biologiques



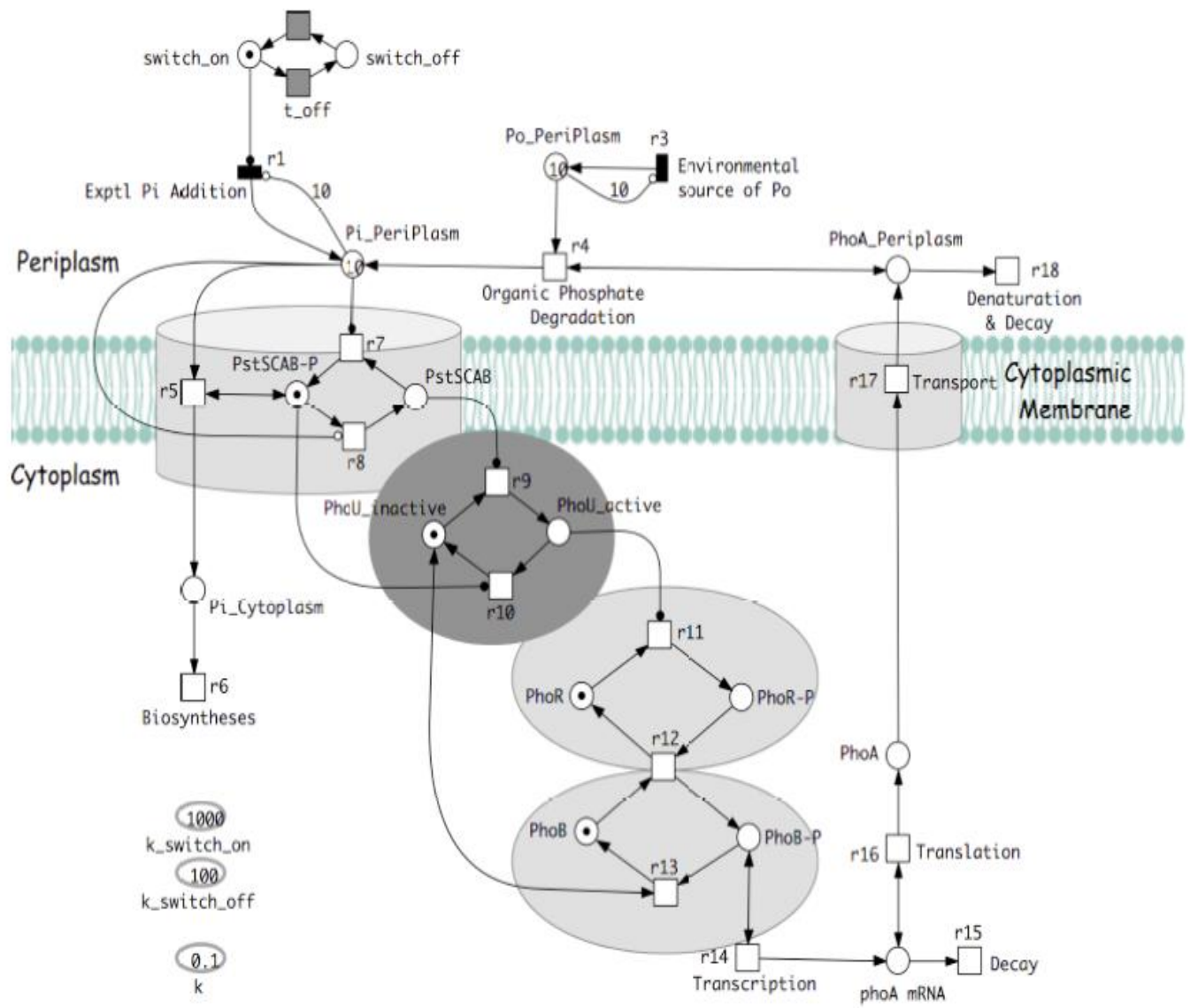
Yeast protein [



from string-db.org

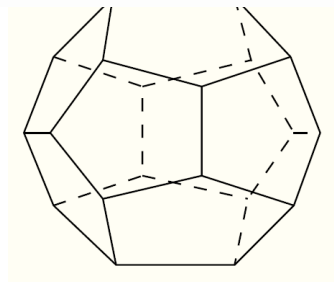
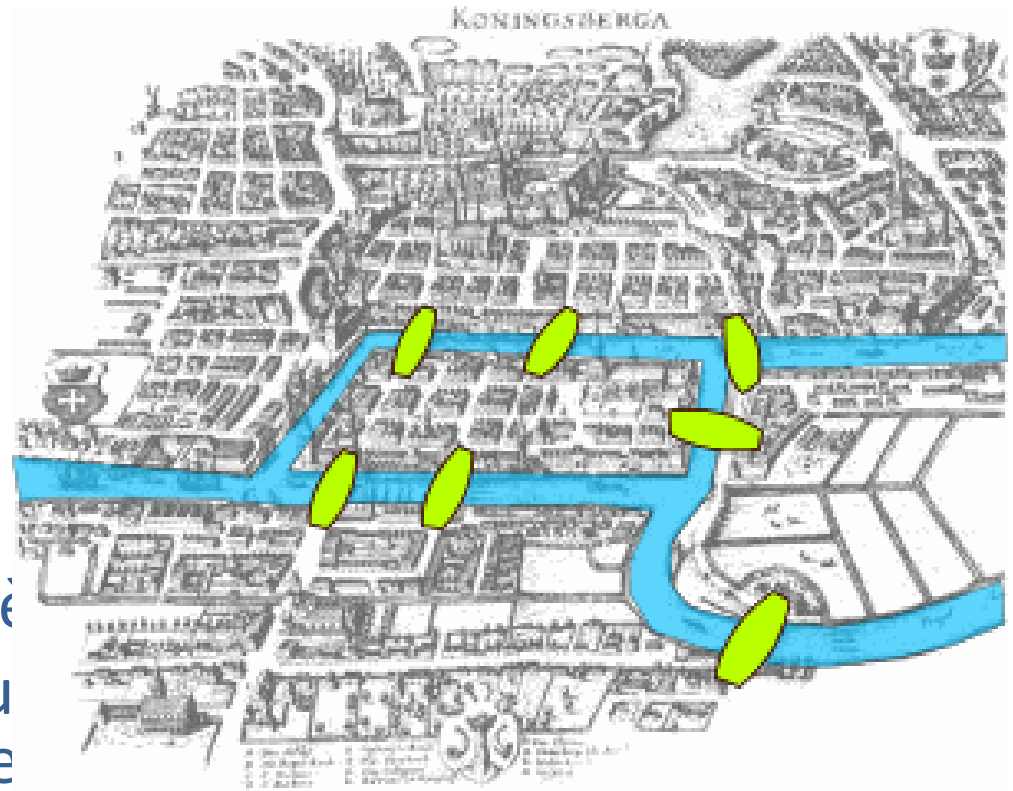


Neidhardt *et al.* 1990



- Historique

- ◆ 1736 Euler
- ◆ 1847 Kirchhof
 - théorie des arbres
 - analyse de circuits électriques
- ◆ 1850 - 1880
 - Cayley, arbres : isométrie
 - Conjecture des 4 couleurs (théorème de Appel)
 - Chemin Hamiltonien
- ◆ 1936 – König 1^{er} ouvrage sur Théorie des graphes
- ◆ 1946 –
 - Kuhn, Ford et Fulkerson, Roy, Claude Berge, ...

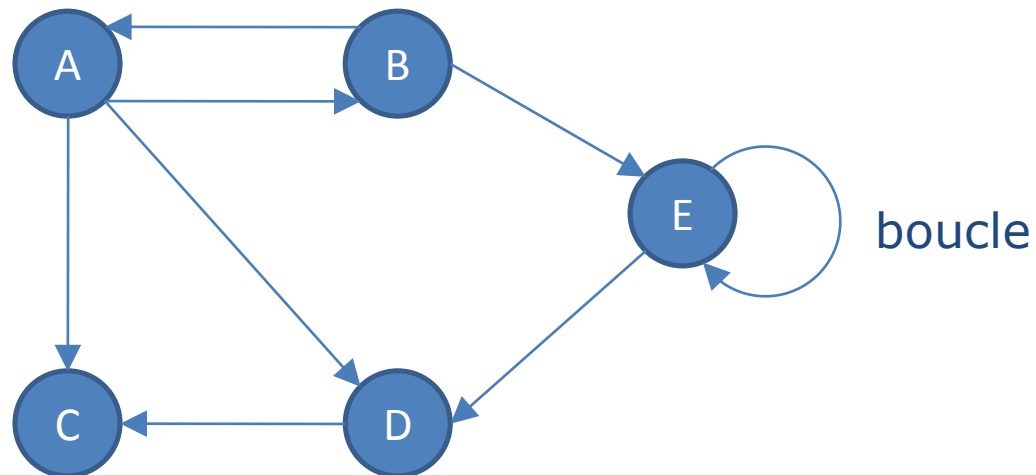


- tournées de distribution, ramassage d'ordure, inspection de réseaux de distribution
- Recherche de chemins, GPS
- tracé automatique (voies métaboliques, circuit imprimé)
- ordonnancement (chantier de construction, processus)
- maximisation de flux (FBA, routier, internet)
- minimisation du coût d'un flot (routage, métabolisme)

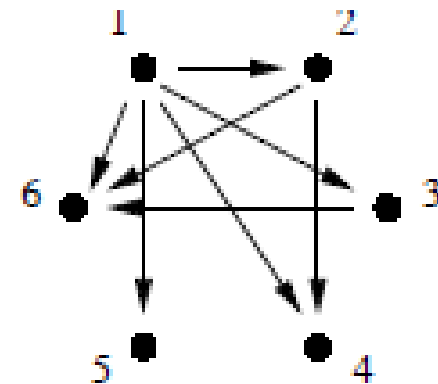
- Réseau métabolique, de régulation de l'expression des gènes
- Réseau d'interactions
- Arbre phylogénétique, de parenté
- Clustering (hiérarchique)
- Ontologies, ex : Gene Ontology
- ...

- Définition
 - ♦ $G = (V, E)$
 - ♦ Ensemble V de **sommets**. Ils peuvent porter des **étiquettes** (nombre entier, couleur, ... peu importe)
 - ♦ Ensemble E d'**arêtes** ou **arcs** de la forme (v_1, v_2) . Les arêtes peuvent être étiquetées (nombre, mot, ...).
 - ♦ Le nombre de sommets est appelé **ordre** du graphe
- Types de graphes
 - ♦ Graphe **non orienté**
 - arêtes : $(v_1, v_2) \leftrightarrow (v_2, v_1)$
 - ♦ Graphe **orienté**
 - arcs : $(v_1, v_2) \neq (v_2, v_1)$
 - ♦ **multigraphe** : possibilité d'avoir plusieurs arêtes reliant les 2 mêmes sommets
 - ♦ **hypergraphe** : une hyperarête peut relier plus de 2 sommets

- $G = (V, E)$
- $V = \{A, B, C, D, E\}$
- $E = \{ (A,B), (B,A), (A,C), (A,D), (B,E), (D,C), (E,D), (E,E) \}$
- Représentation **sagittale** :



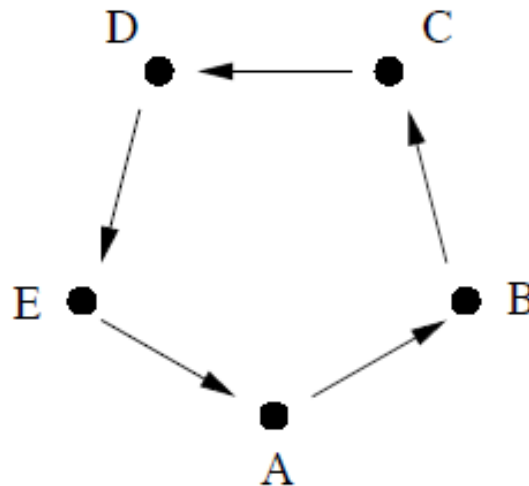
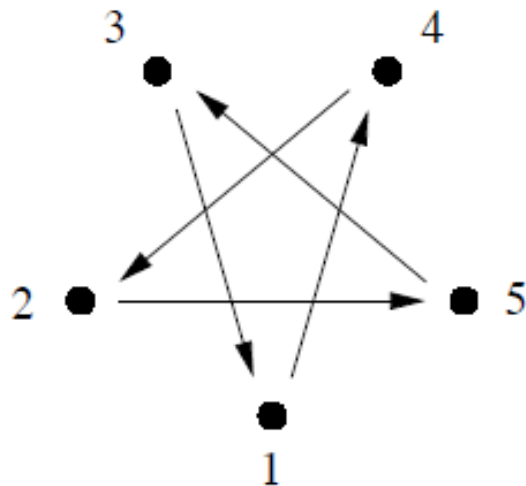
- $G = (V, E)$
- un arc $a = (x, y) \in E$ peut être noté $x \rightarrow y$
- x et y sont les **extrémités** de a
- x est le **début** ou **origine** ou **extrémité initiale** de a . y est la **fin** ou l'**extrémité finale** de a .
- a est **sortant** en x et **incident** en y .
- x et y sont **adjacents**.
- y est un **successeur** de x .
- x est un **prédécesseur** de y .
- propriété caractéristique d'un graphe
 - ♦ ex: $\forall x, y \in V, (x, y) \in E \iff x \text{ divise } y$



- 2 graphes orientés $G1 = (V1, E1)$ et $G2 = (V2, E2)$ sont **isomorphes** s'il existe une application bijective $f : V1 \rightarrow V2$ telle que

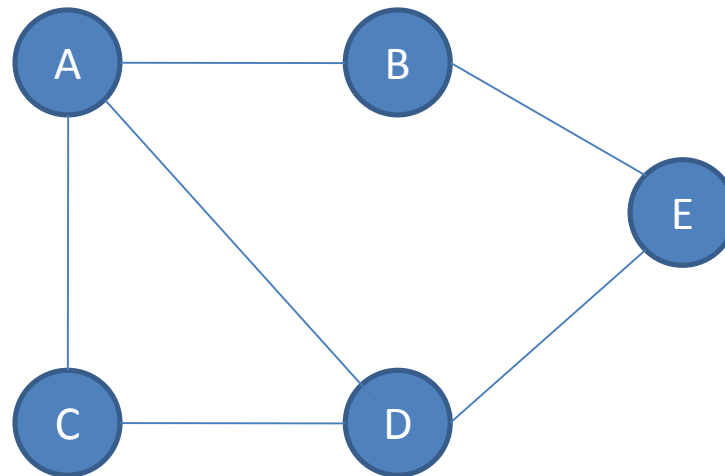
$$\forall x, y \in V1 / (x, y) \in E1 \Leftrightarrow (f(x), f(y)) \in E2$$

- Exemple :



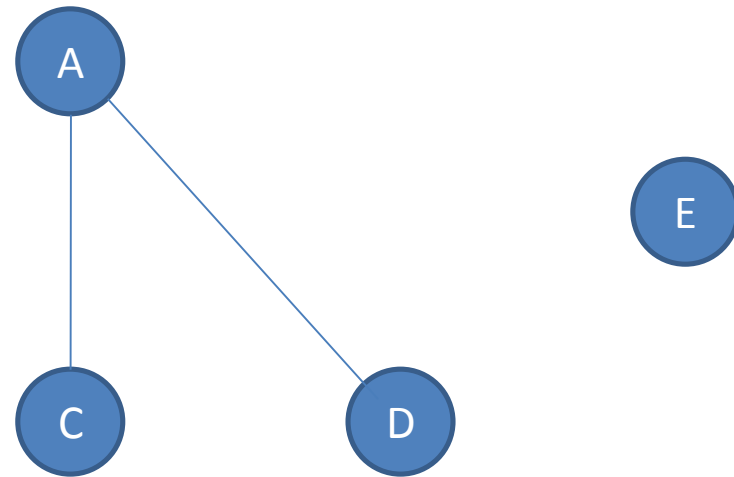
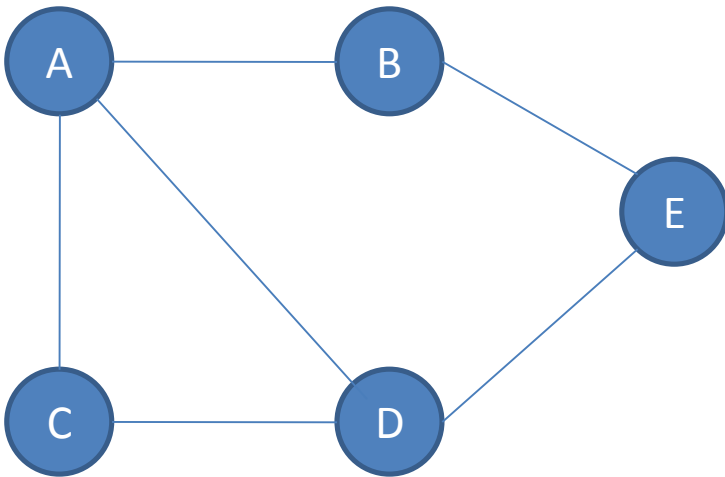
$$f : \begin{cases} 1 \mapsto A \\ 2 \mapsto C \\ 3 \mapsto E \\ 4 \mapsto B \\ 5 \mapsto D \end{cases}$$

- $G = (V, E)$
- $V = \{A, B, C, D, E\}$
- $E = \{ (A,B), (A,C), (A,D), (B,E), (C,D), (D,E) \}$
- Représentation sagittale :



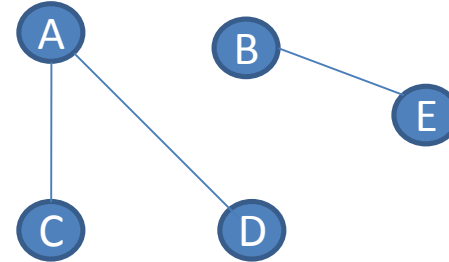
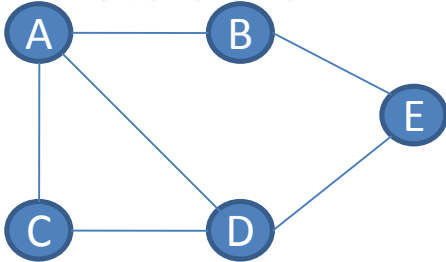
- On peut associer à un graphe orienté un graphe non orienté appelé **graphe non orienté associé** ou **sous-jacent**

- Soit $G = (V, E)$ un graphe (orienté ou non). Un **sous-graphe** de G est un graphe $G' = (V', E')$ tel que $V' \subseteq V$ et $E' \subseteq E$

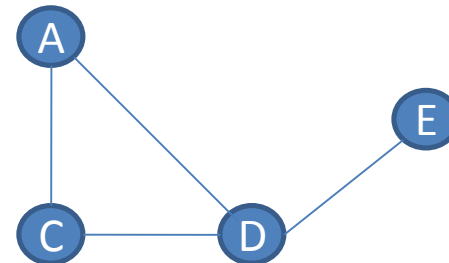
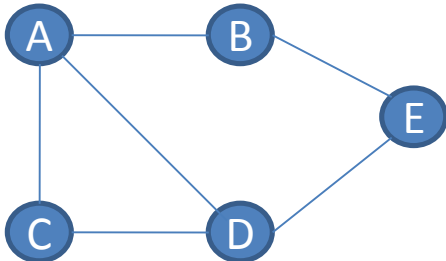


- Exemple : voie métabolique \subset réseau métabolique

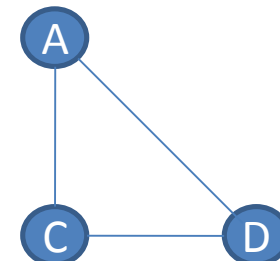
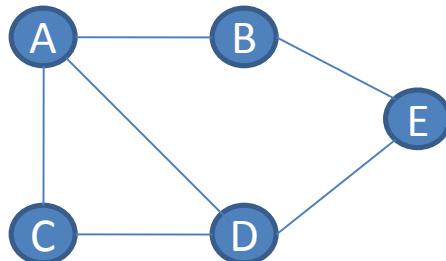
- Un sous-graphe G' d'un graphe G est **couvrant** si il contient tous les sommets de G



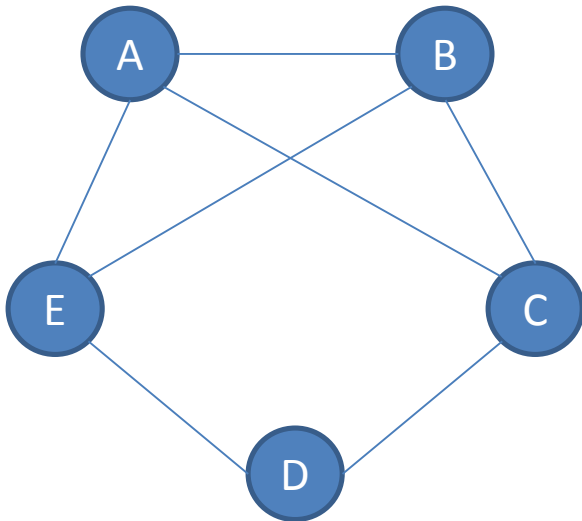
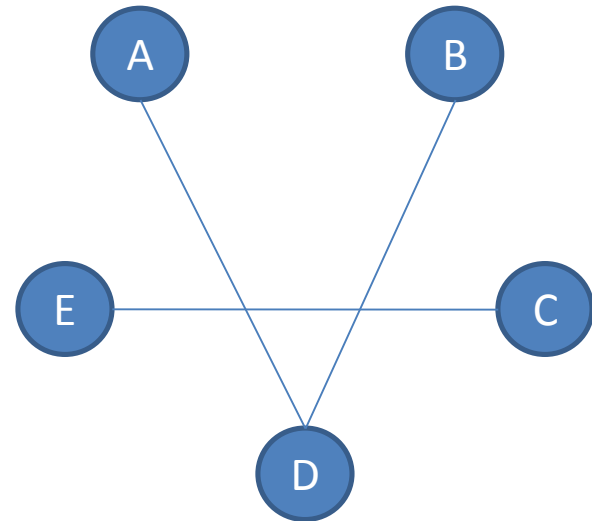
- Un sous-graphe G' d'un graphe G est un **sous-graphe induit** si E' est formé de tous les arcs (ou arêtes) de G ayant leurs extrémités dans G'



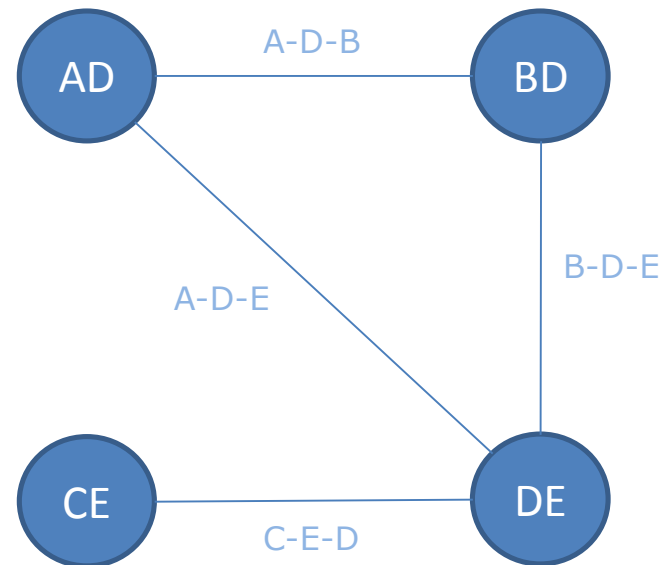
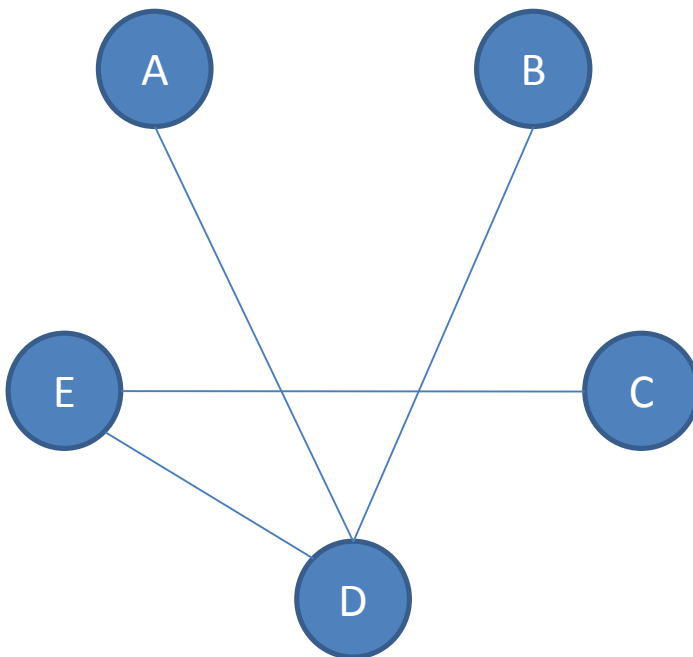
- Une **clique** est un sous-graphe induit complet



- Le **complémentaire** ou **complément** ou **inverse** d'un graphe G est noté \overline{G} , il a les mêmes sommets qui sont reliés si et seulement si ils ne sont pas reliés dans G

 G  \overline{G}

- Le line graph d'un graphe G est le graphe $L(G)$ dans lequel sont inversés sommets et arêtes, c'est-à-dire que **deux sommets adjacents** dans le line graph **correspondent à deux arêtes incidentes à un même sommet** dans G .

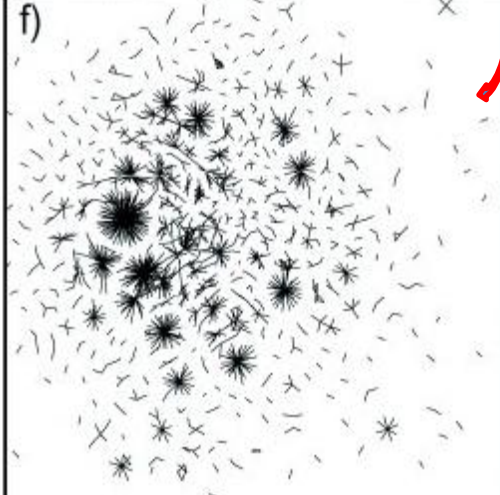
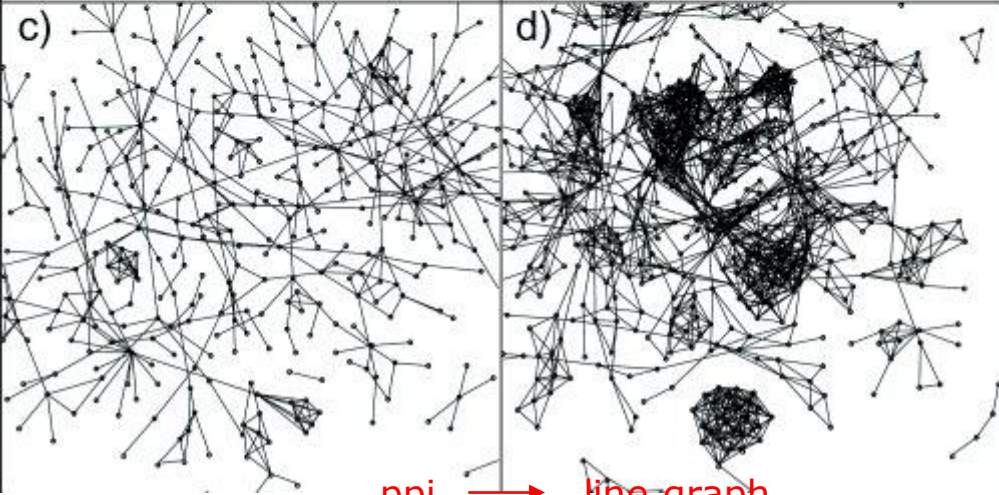
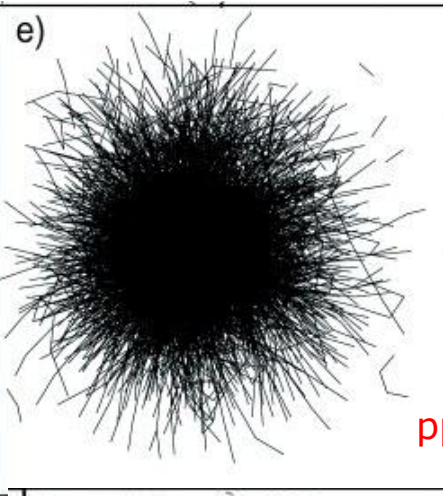
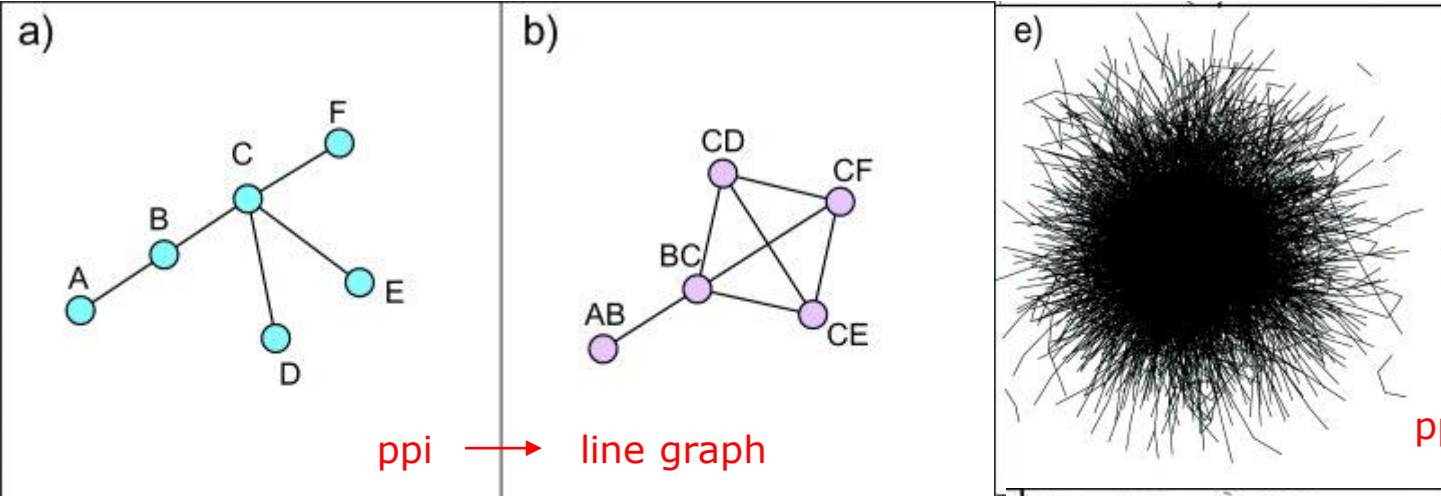


Detection of Functional Modules From Protein Interaction Networks

Jose B. Pereira-Leal,¹ Anton J. Enright,² and Christos A. Ouzounis^{1*}

¹Computational Genomics Group, The European Bioinformatics Institute, Cambridge, United Kingdom

²Computational Biology Center, Memorial Sloan-Kettering Cancer Center, New York, New York



- Pour un graphe orienté :
 - ♦ degré entrant : nombre de prédécesseurs d'un sommet, noté $d_-(x)$
 - ♦ degré sortant : nombre de successeurs d'un sommet, noté $d_+(x)$
 - ♦ degré total : nombre d'arcs dont x est le début ou la fin (on compte donc 2 fois les boucles), $d(x) = d_-(x) + d_+(x)$

$$\sum_{x \in V} d_-(x) = \sum_{x \in V} d_+(x) = |E| \qquad \sum_{x \in V} d(x) = 2|E|$$

- Un sommet de degré entrant non nul et de degré sortant nul est un **puit**.
- Un sommet de degré entrant nul et de degré sortant non nul est appelé **source**.
- Un sommet de degré nul est un sommet **isolé**.
- Pour un graphe non orienté : $d(x)$

- Pour un graphe orienté
 - ♦ Un **chemin** C est une suite $(x_0, x_1, \dots, x_{n-1}, x_n)$ de sommets de G tel que 2 sommets consécutifs quelconque x_i et x_{i+1} sont reliés par un arc de G
 - ♦ x_0 et x_n sont le début et la fin du chemin C
 - ♦ C est de longueur n (nombre d'arcs)
- Pour un graphe non orienté, on parle de **chaîne**

- Un **circuit** dans un graphe orienté est un chemin de longueur non nulle dont le début et la fin sont identiques.
- Un **chemin simple** ne passe pas 2 fois par le même arc.
- Un **chemin élémentaire** ne passe pas 2 fois par le même sommet.
- Pour un graphe non orienté on parle de **cycle**.

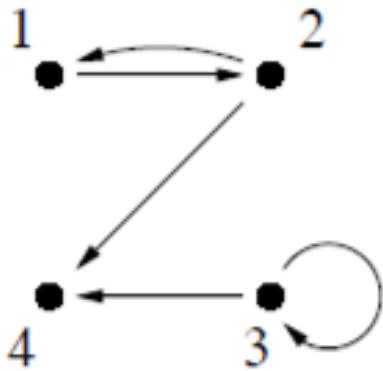
- Un graphe non orienté est **connexe** si pour tout couple de sommets x, y , il existe une chaîne reliant x à y .
- Une **composante connexe** d'un graphe est un sous ensemble **maximal** de sommets dont toutes les paires de sommets sont reliées par une chaîne.
- Les composantes connexes d'un graphe forment une **partition** du graphe.
- Un graphe orienté est **connexe** si son graphe non orienté associé est connexe.
- Un graphe orienté est **fortement connexe** si pour toute paire de sommets, il existe un chemin les reliant.
- Un **point d'articulation** est un sommet dont la suppression augmente le nombre de composantes connexes.
- Un **isthme** est une arête dont la suppression augmente le nombre de composantes connexes.
- Un **ensemble d'articulation** d'un graphe connexe est un ensemble de sommets dont la suppression rend le graphe non connexe.
- Un **pont** dans un graphe connexe est une arête dont la suppression déconnecte le graphe.

- Pour un graphe non orienté, une chaîne (respectivement un cycle) eulérienne passe une et une seule fois par toutes les arêtes du graphe.
- Pour un graphe orienté, un chemin (resp. un circuit) eulérien passe une et une seule fois par tous les arcs du graphe.
- Pour un graphe non orienté, une chaîne (respectivement un cycle) hamiltonienne passe une et une seule fois par tous les sommets du graphe.
- Pour un graphe orienté, un chemin (resp. un circuit) hamiltonien passe une et une seule fois par tous les sommets du graphe.

- $G = (V, E)$
- Les sommets sont numérotés de 1 à n ; n étant l'ordre du graphe.
- La matrice d'adjacence de G est la matrice carrée M définie par

$$m_{ij} = \begin{cases} 1 & \text{si } (i, j) \in E \\ 0 & \text{sinon} \end{cases}$$

- Exemple :



$$M = \begin{matrix} & \xrightarrow{j} & & & & \\ \begin{matrix} i \\ \downarrow \end{matrix} & \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} & & & & \\ & & & & & d_+ = \sum M_{i\bullet} \\ & & & & & \\ & & & & & d_- = \sum M_{\bullet j} \end{matrix}$$

- $G = (V, E)$

- Exemple :

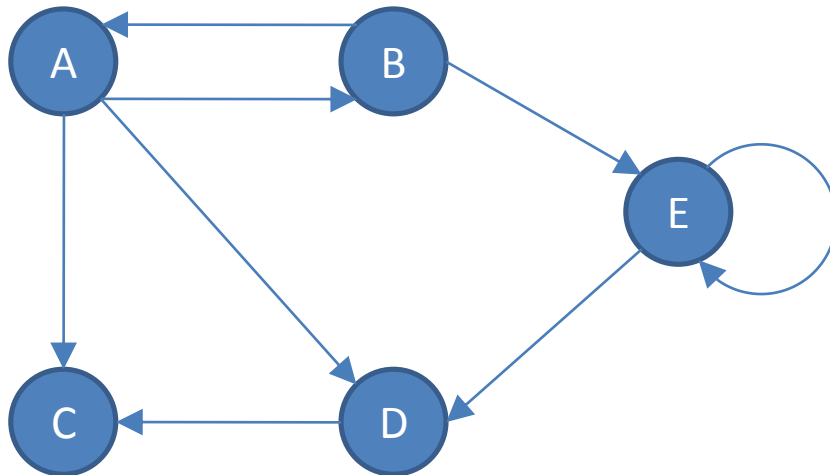
A (B, C, D)

B (A, E)

C

D (C)

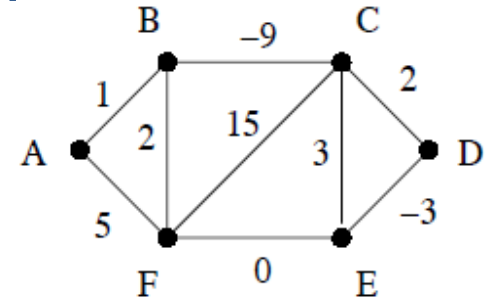
E (D, E)



- Un graphe **valué** est muni d'une application attribuant une valeur à chaque arc ou arête appelée **valuation**.
- Notation : graphe $G = (S,A)$

$$v : A \rightarrow \mathbb{R}$$

$$(x,y) \mapsto v(x,y)$$



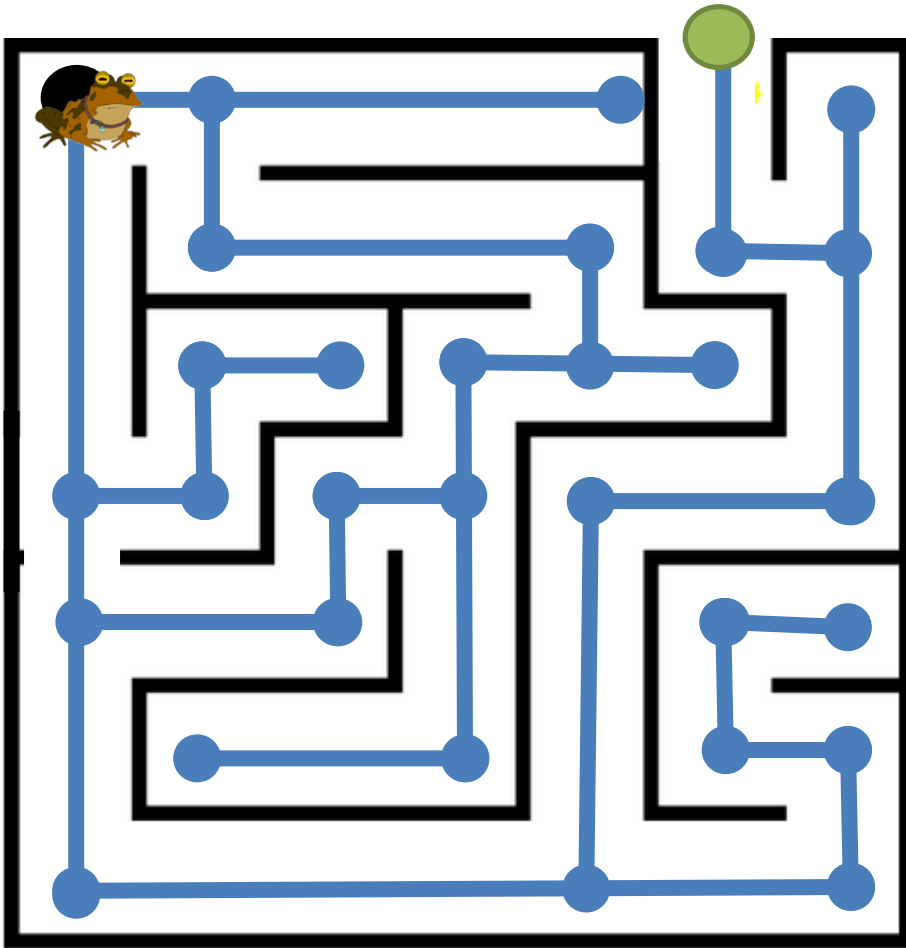
- Représentation matricielle : $G = (S,A,v)$

$$m_{ij} = \begin{cases} v(i,j) & \text{si } (i,j) \in A \\ +\infty & \text{sinon} \end{cases}$$

$$\begin{pmatrix} \infty & 1 & \infty & \infty & \infty & 5 \\ 1 & \infty & -9 & \infty & \infty & 2 \\ \infty & -9 & \infty & 2 & 3 & 15 \\ \infty & \infty & 2 & \infty & -3 & \infty \\ \infty & \infty & 3 & -3 & \infty & 0 \\ 5 & 2 & 15 & \infty & 0 & \infty \end{pmatrix}$$

- La **valuation** ou **longueur** d'un chemin (ou d'une chaîne) est la somme des valuations de chacun des arcs (ou arêtes) qui le composent.

ex: la valuation de (A,F,C,E,D) est $5 + 15 + 3 - 3 = 20$



- Existe-t-il un chemin ou une chaîne ?
- Détection de cycles/circuits
- Tri topologique
- Composantes (fortement) connexes
- Plus courts chemins

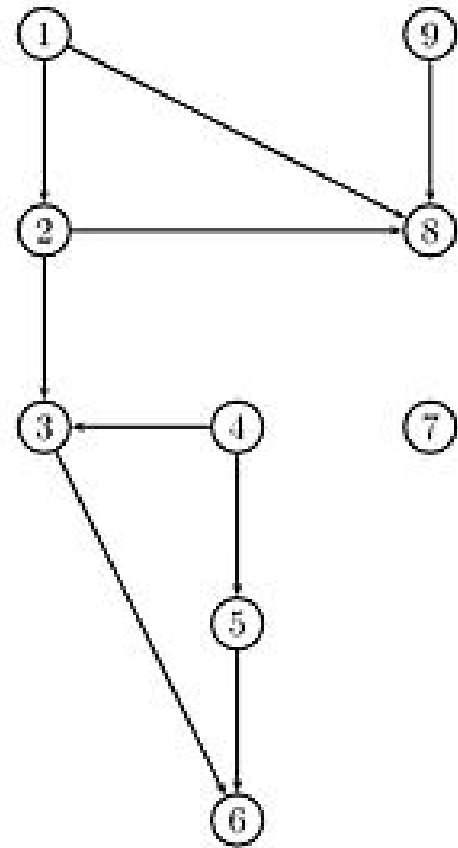
DFS (graphe **G**, sommet **s**)

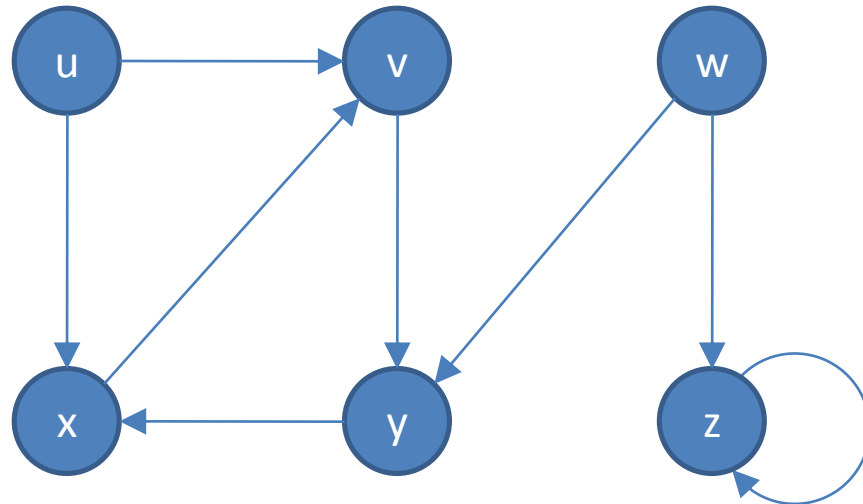
Marquer(**s**)

pour chaque successeur de **s faire**

si *NonMarqué*(**successeur**) **alors**

DFS(**G**, **successeur**)





DFS (G)

```

1  for each vertex  $u \in V [G]$ 
2      do  $color[u] \leftarrow WHITE$ 
3           $\pi[u] \leftarrow NIL$ 
4   $time \leftarrow 0$ 
5  for each vertex  $u \in V [G]$ 
6      do if  $color[u] = WHITE$ 
7          then DFS-VISIT( $u$ )

```

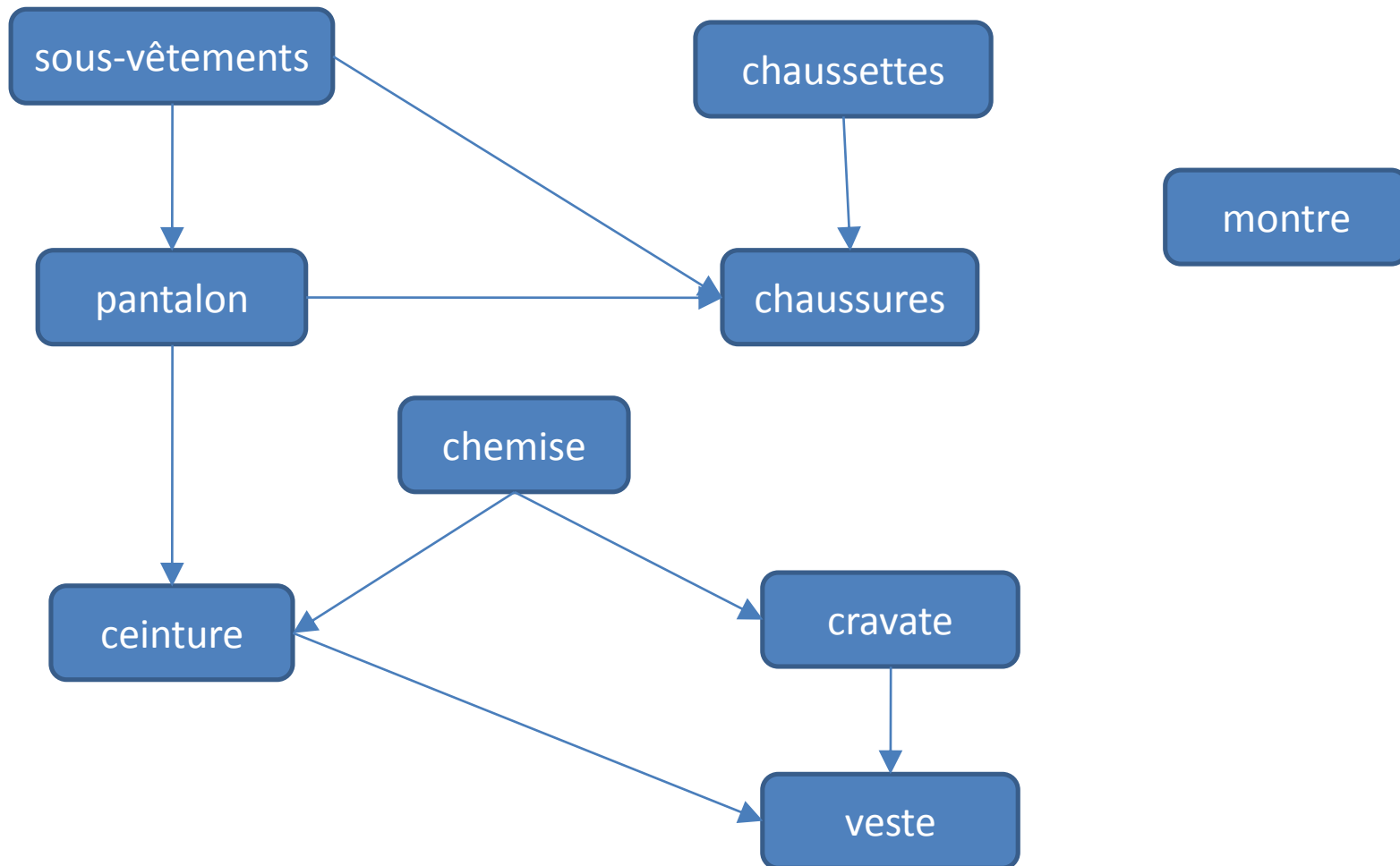
DFS-VISIT(u)

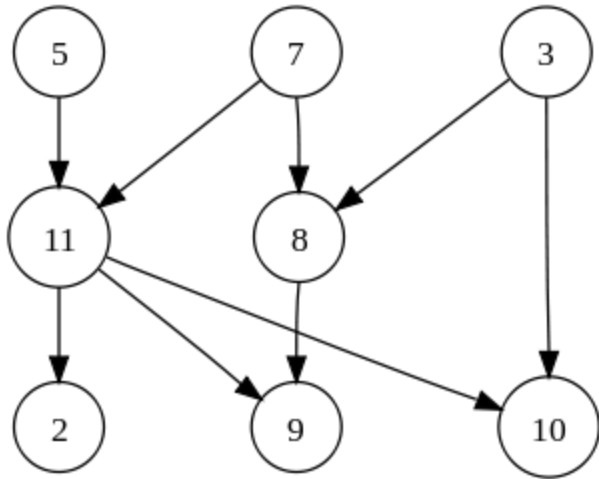
```

1   $color[u] \leftarrow GRAY$        $\triangleright$ White vertex  $u$  has just been discovered.
2   $time \leftarrow time + 1$ 
3   $d[u] \leftarrow time$ 
4  for each  $v \in Adj[u]$   $\triangleright$ Explore edge( $u, v$ ).
5      do if  $color[v] = WHITE$ 
6          then  $\pi[v] \leftarrow u$ 
7                  DFS-VISIT( $v$ )
8   $color[u] \leftarrow BLACK$        $\triangleright$ Blacken  $u$ ; it is finished.
9   $f[u] \leftarrow time \leftarrow time + 1$ 

```

Grphe orienté, DFS, tri topologique





tsort.in

```

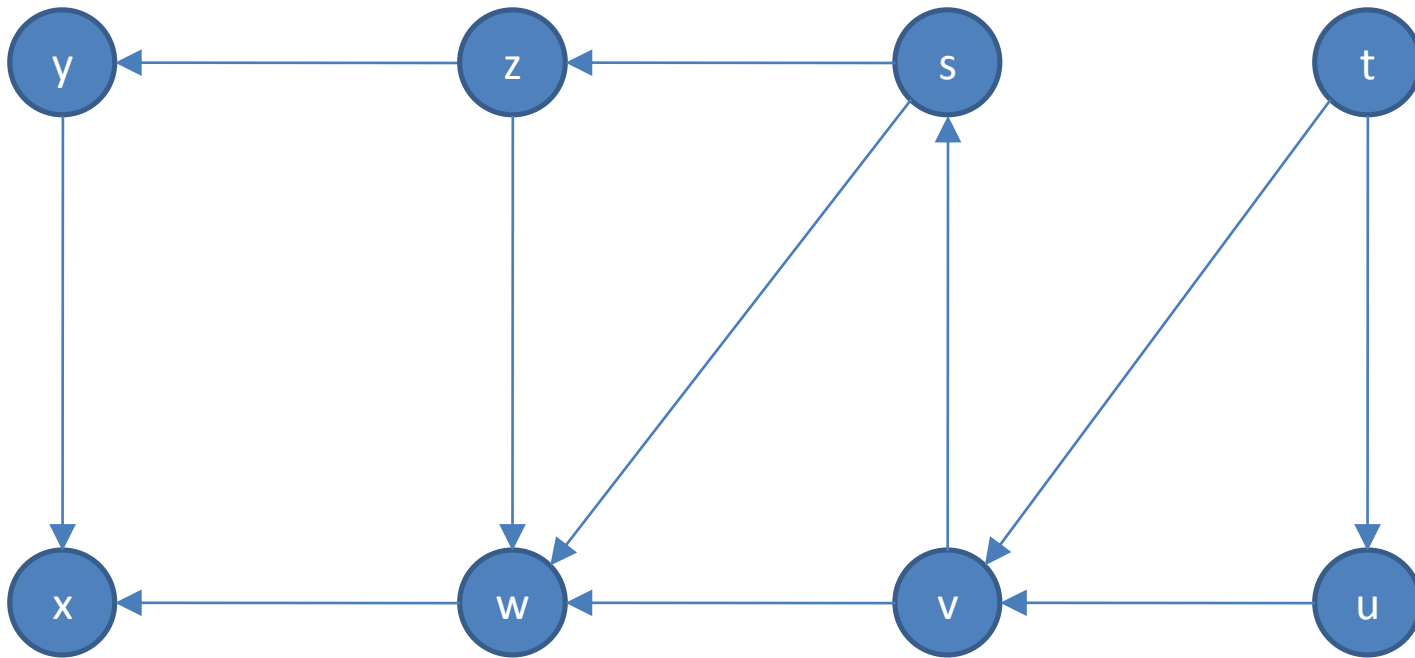
3 8
3 10
5 11
7 8
7 11
8 9
11 2
11 9
11 10
  
```

- commande linux : `tsort`

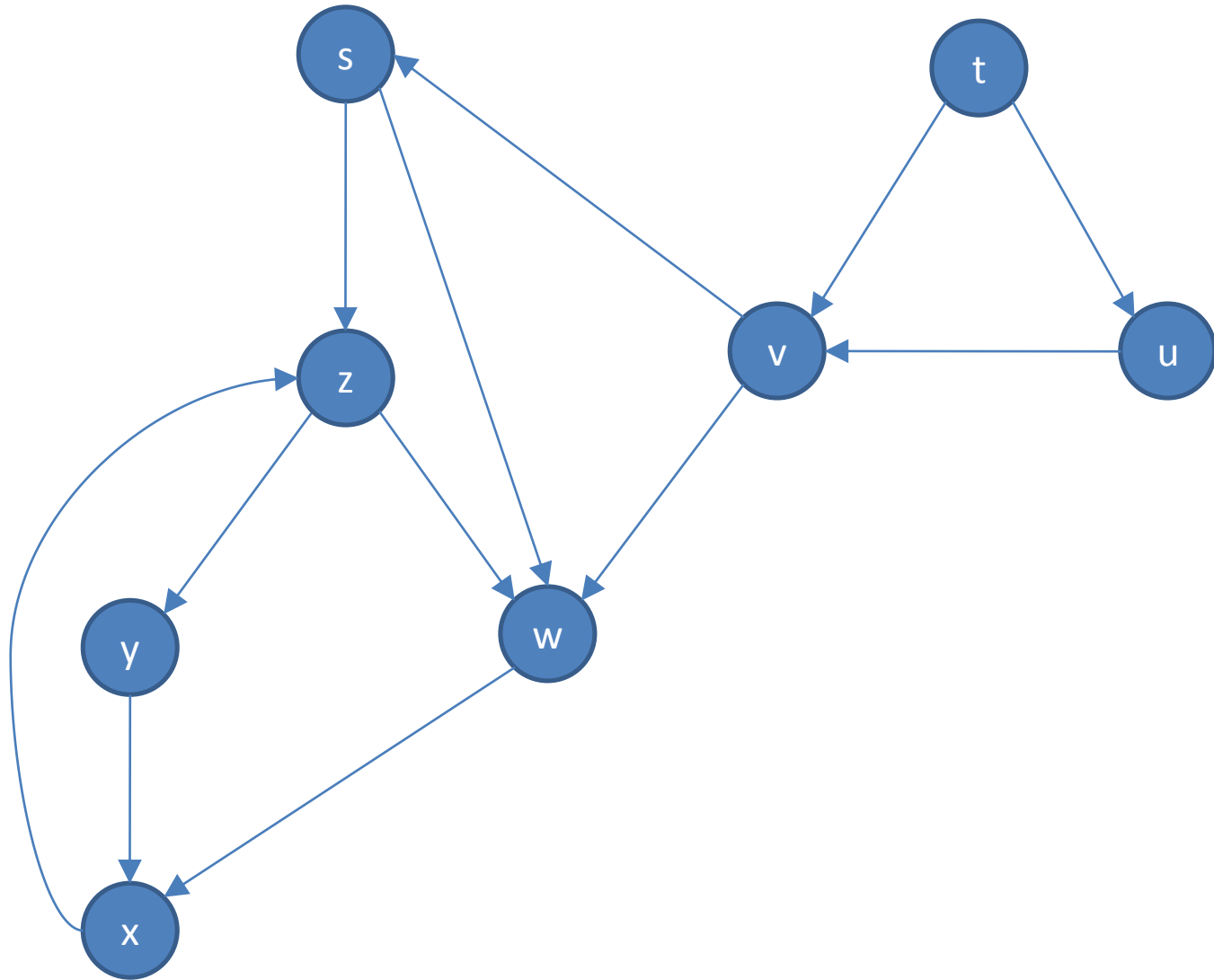
```

3
5
7
11
8
10
2
9
  
```

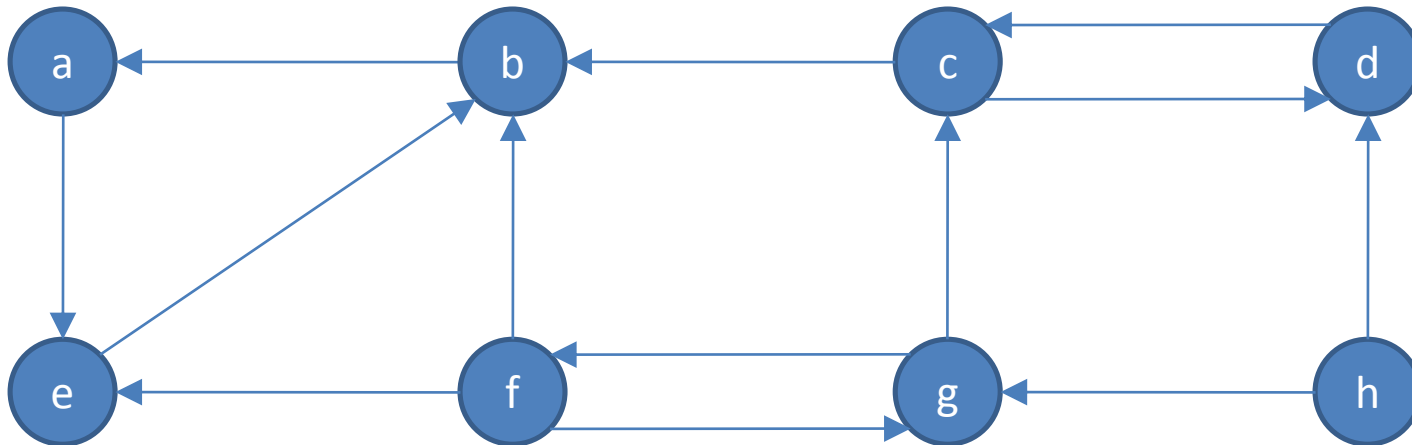
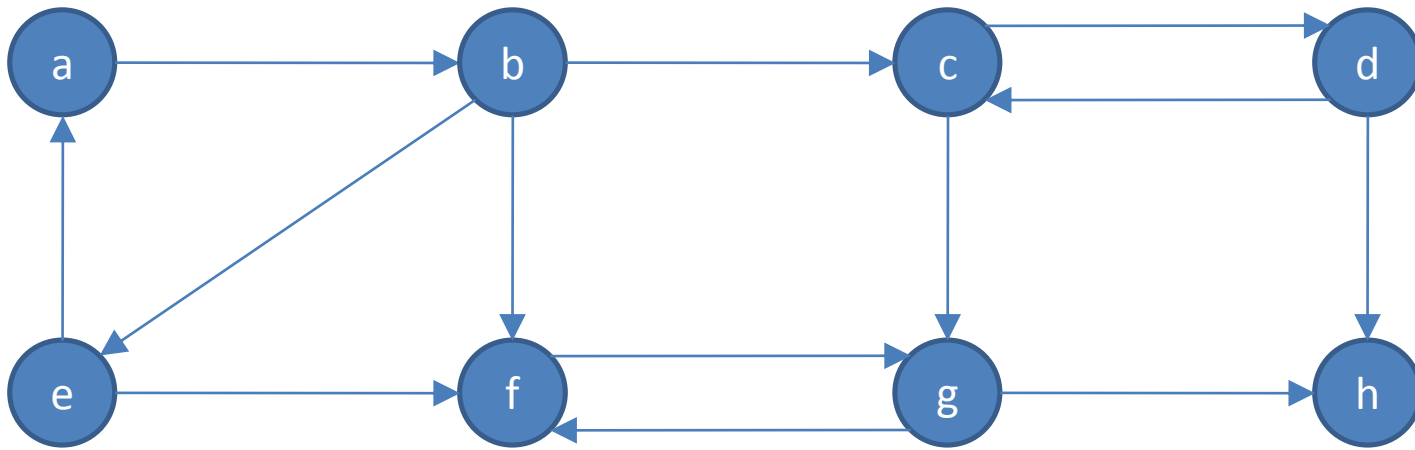
Grphe orienté, tri topologique



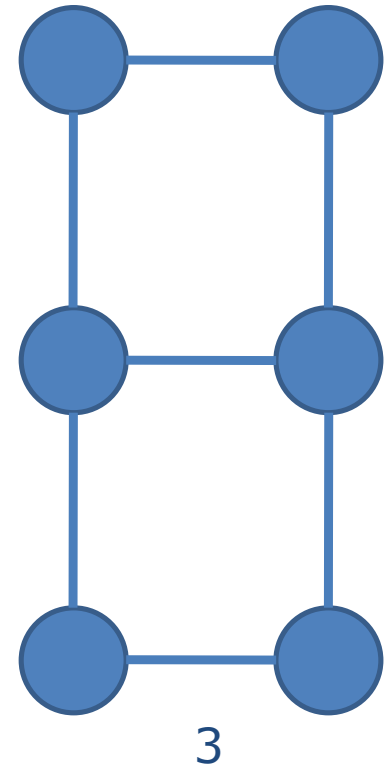
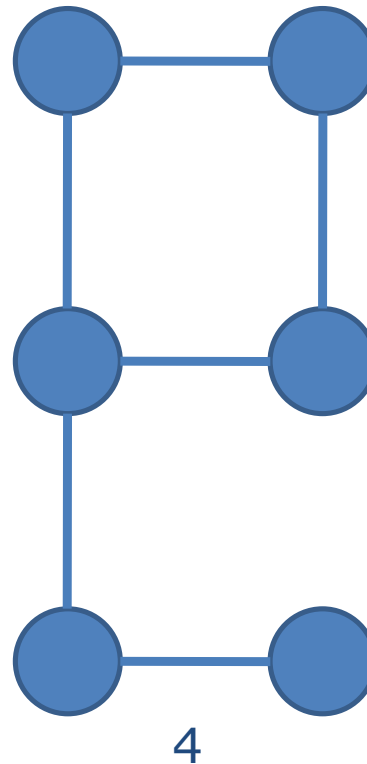
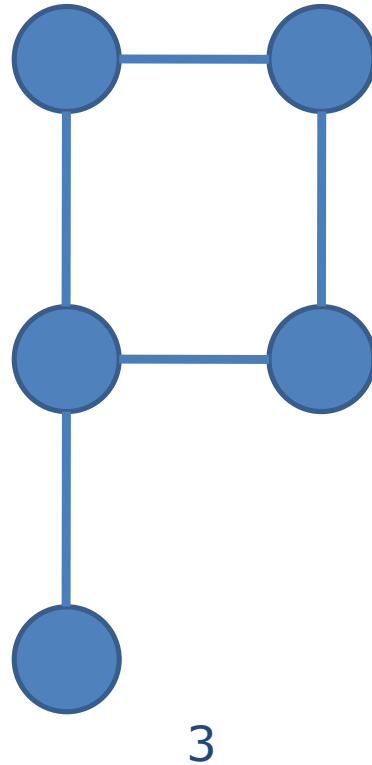
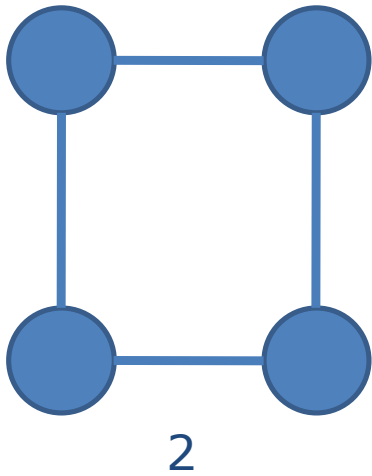
Grphe orienté



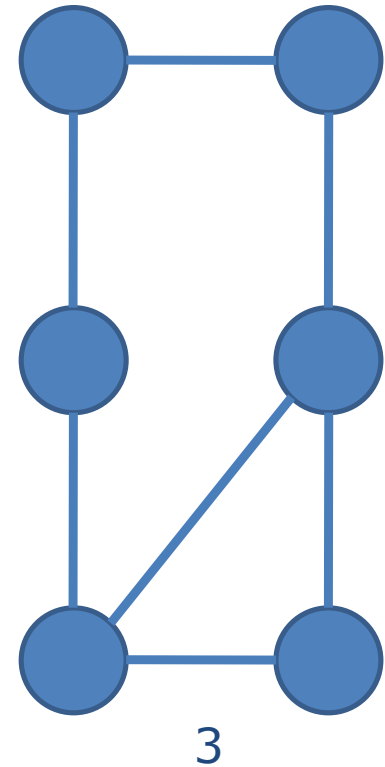
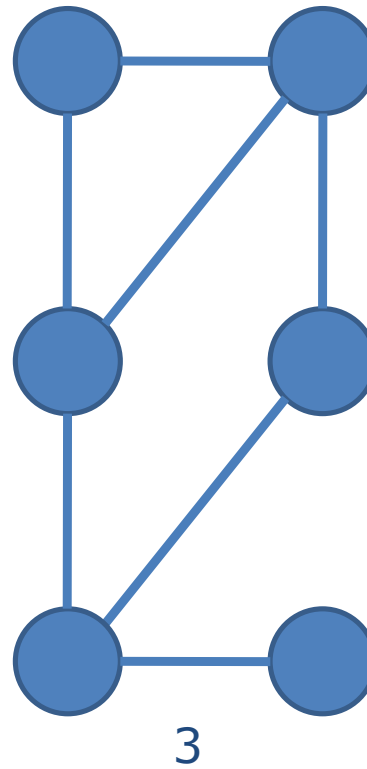
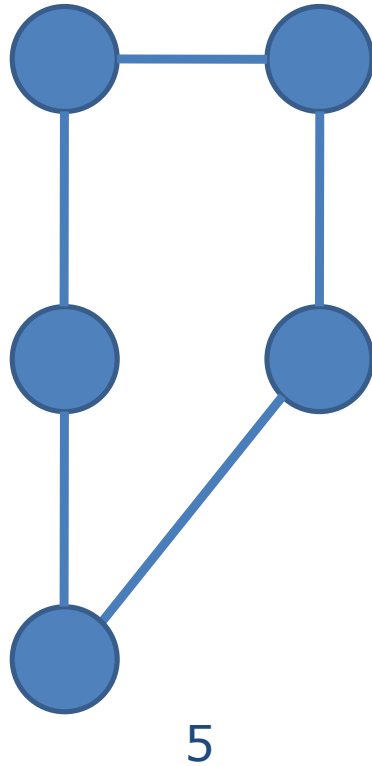
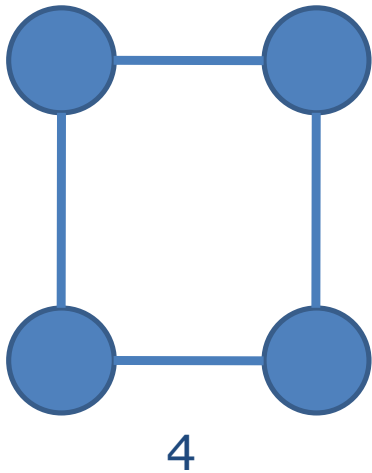
Grphe orienté, composantes fortement connexes



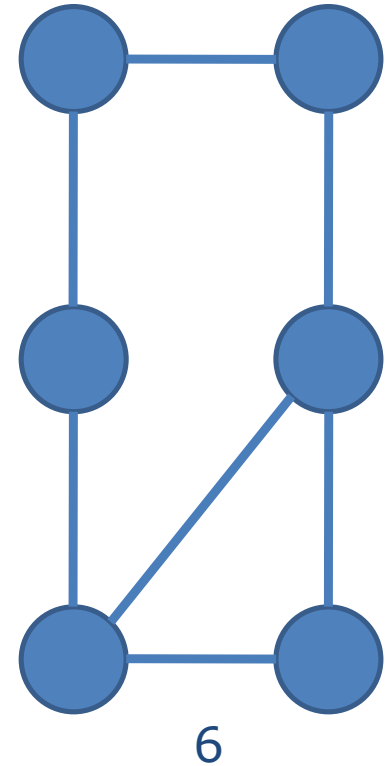
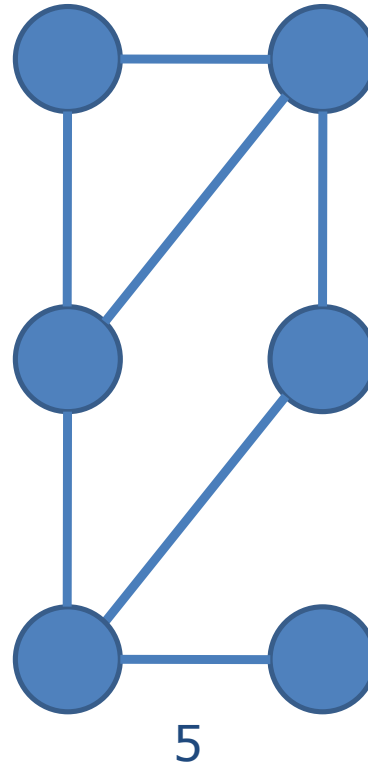
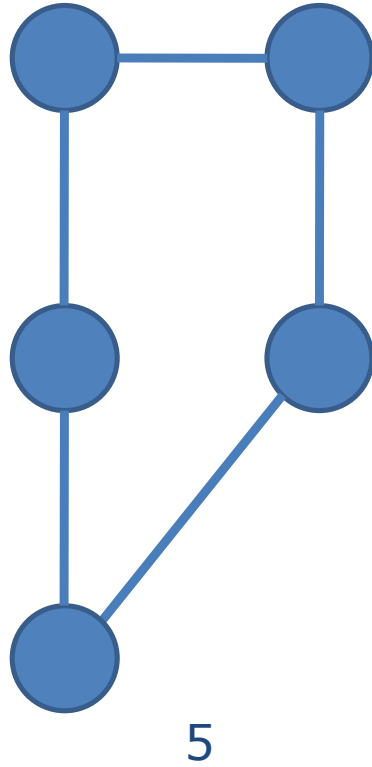
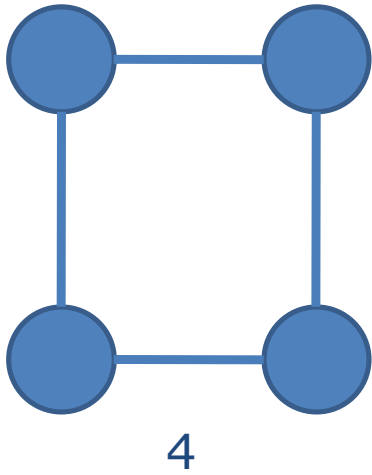
- **diamètre** : La longueur du plus court chemin entre les sommets les plus distants d'un graphe d mesure l'étendue d'un graphe, la longueur topologique entre deux sommets.
- **maille** : longueur du plus court cycle. Infinie pour un graphe acyclique.
- **circonférence** : longueur du plus grand cycle.
- **nombre cyclomatique** : indique le nombre de cycles que possède un réseau.
 - ♦ $u = e - v + p$
 - ♦ e : nombre d'arêtes, v : nombre de sommets et p : nombre de composantes connexes.
- Indice de **Pi** : Indique la relation entre la longueur totale du graphe $L(G)$ et la distance le long du diamètre $D(G)$. Un index élevé pointe vers un réseau développé. C'est une mesure de distance par unité de diamètre et un indicateur de la forme d'un réseau.
- Indice d'**Eta** : Indique la longueur moyenne des arcs. L'ajout de nouveaux sommets provoquera une décroissance d'Eta.
- Indice de **Beta** : Indique la relation entre le nombre d'arcs (e) sur le nombre de sommets (v).
- Indice **Gamma** (g) : Relation entre le nombre d'arcs observés et le nombre d'arcs possibles. La valeur est entre 0 et 1. Une valeur de 1 indique un réseau complètement connecté.
- **coefficient d'agglomération** : nombre d'arêtes entre les voisins / nombre d'arêtes possibles
- **betweenness** : Nombre de plus courts chemins passant par un(e) sommet/arête



diamètre : La longueur du plus court chemin entre les sommets les plus distants d'un graphe d mesure l'étendue d'un graphe, la longueur topologique entre deux sommets

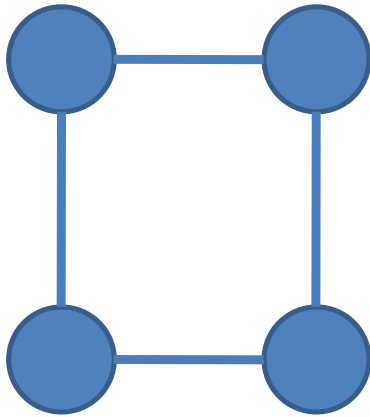


maille : longueur du plus court cycle.
Infinie pour un graphe acyclique.

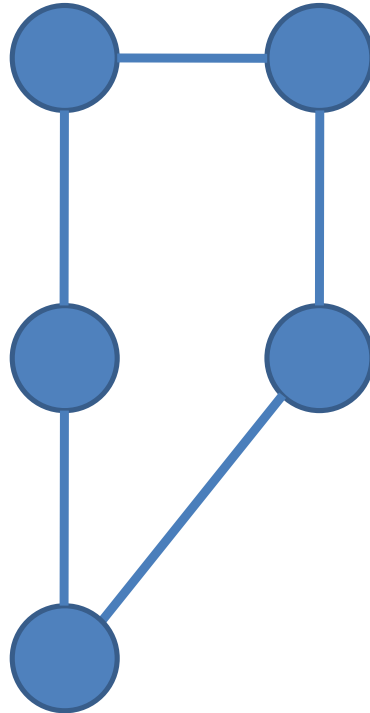


circonférence : longueur du plus grand cycle.

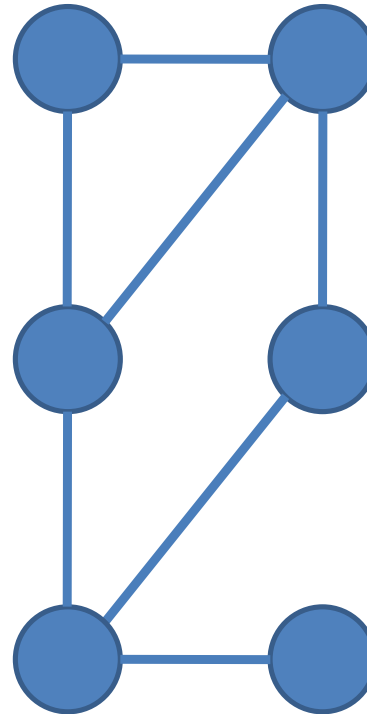
Nombre cyclomatique



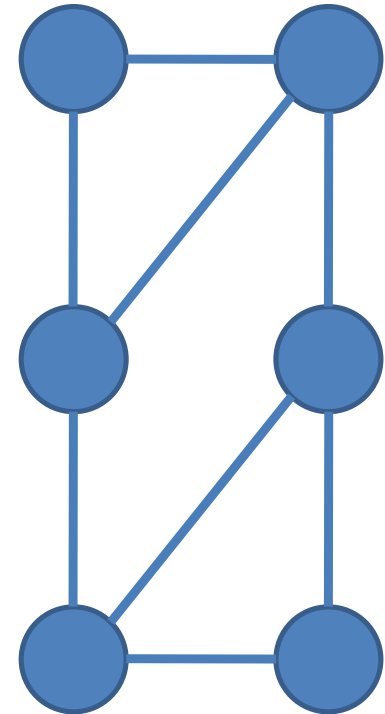
$$4 - 4 + 1 = 1$$



$$5 - 5 + 1 = 1$$



$$7 - 6 + 1 = 2$$



$$8 - 6 + 1 = 3$$

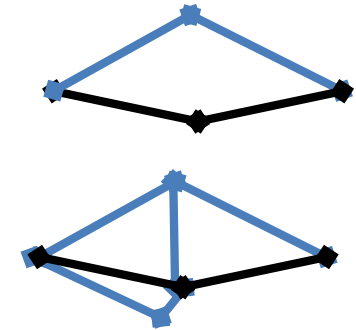
nombre cyclomatique : indique le nombre de cycles que possède un réseau.

$$u = e - v + p$$

e : nombre d'arêtes, v : nombre de sommets et p : nombre de composantes connexes.

- Indice de P_i : Un index élevé pointe vers un réseau développé.

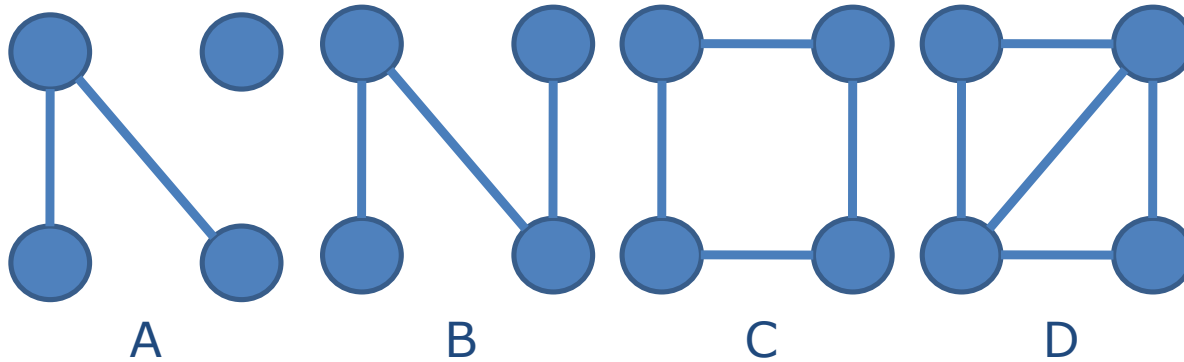
longueur : $L(G)$	diamètre : $D(G)$	$P_i = L(G)/D(G)$
130 km	75 km	1.733
245 km	75 km	3.266



- Indice d'Eta : L'ajout de nouveaux sommets provoque une décroissance.

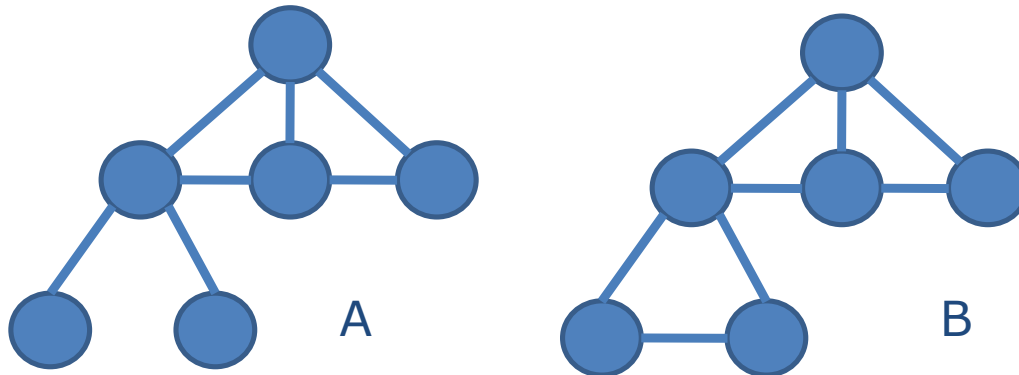
$L(G)$	Nombre d'arcs	$\text{Eta} = L(G)/ E $
130 km	15 arcs	8.666 km/link
130 km	25 arcs	5.3 km/link

- Indice **Beta** : Indique la relation entre le nombre d'arcs (e) sur le nombre de sommets (v).



	e	v	Beta
A	2	4	0.5
B	3	4	0.75
C	4	4	1.0
D	5	4	1.25

- Indice **Gamma** (g) : Relation entre le nombre d'arcs observés et le nombre d'arcs possibles. La valeur est entre 0 et 1. Une valeur de 1 indique un réseau complètement connecté.



	e	$1/2v(v-1)$	g
A	7	15	0.466
B	8	15	0.533

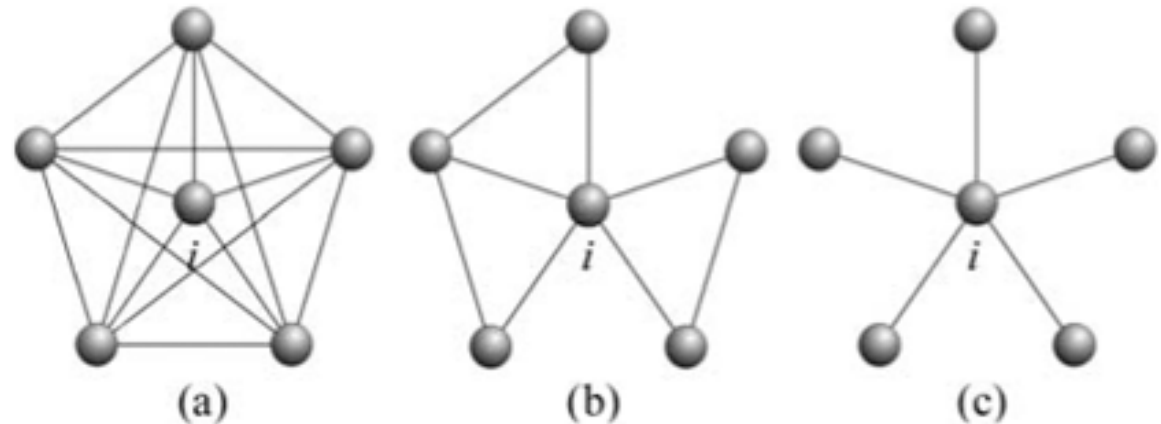
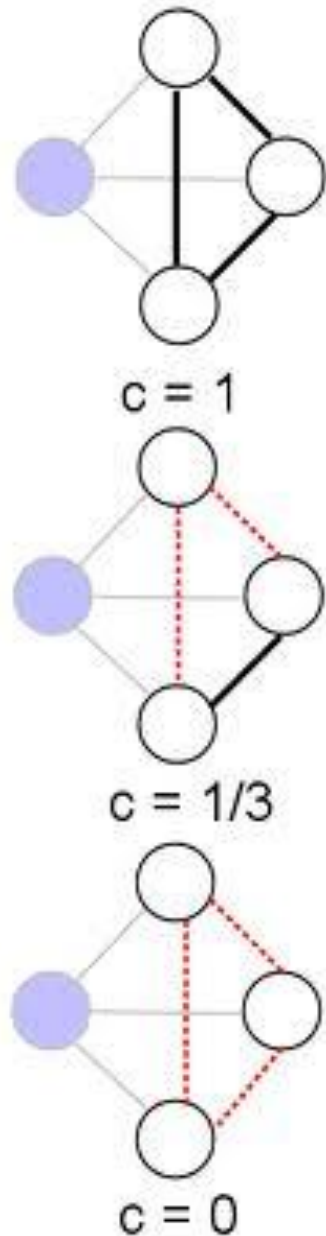
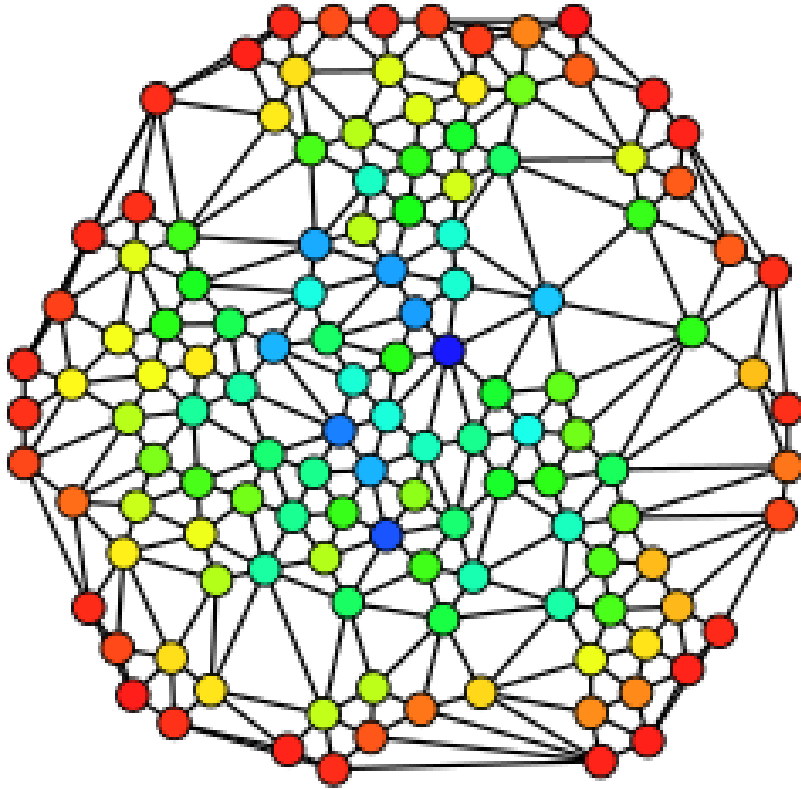


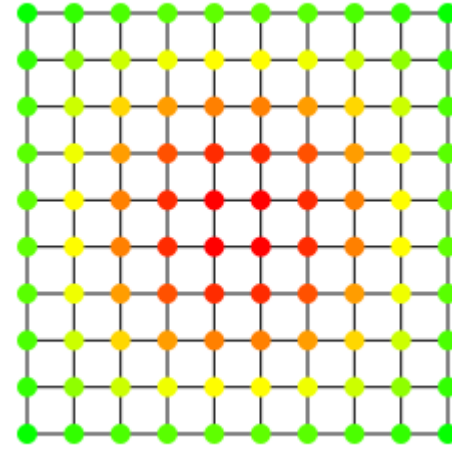
Figure 4 - Example of three networks and respective clustering coefficients (see Eq. (1)). In (a), $cc_i = \frac{10(2)}{5(4)} = 1$ (the vertices around i are fully connected), (b) $cc_i = \frac{3(2)}{5(4)} = 0.3$ and (c) $cc_i = \frac{0(2)}{5(4)} = 0$. The maximum number of edges among the neighbors of i is given by $k_i(k_i - 1)/2$.

source : Costa *et al.*, 2008

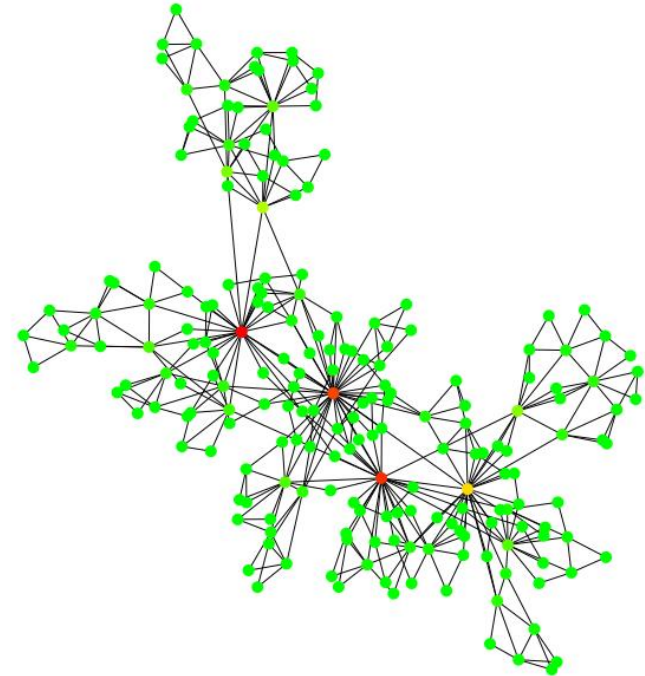
Betweenness



source : Wikipedia

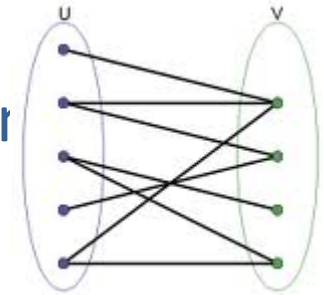


source : graphstream-project.org

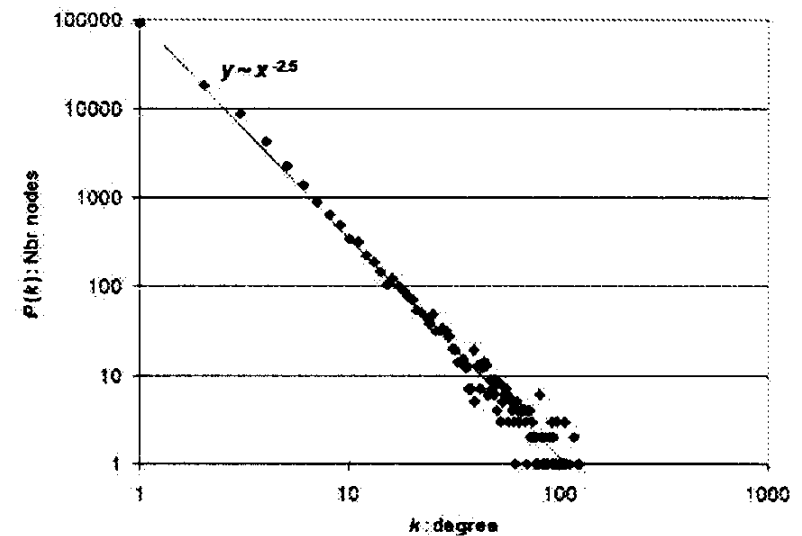
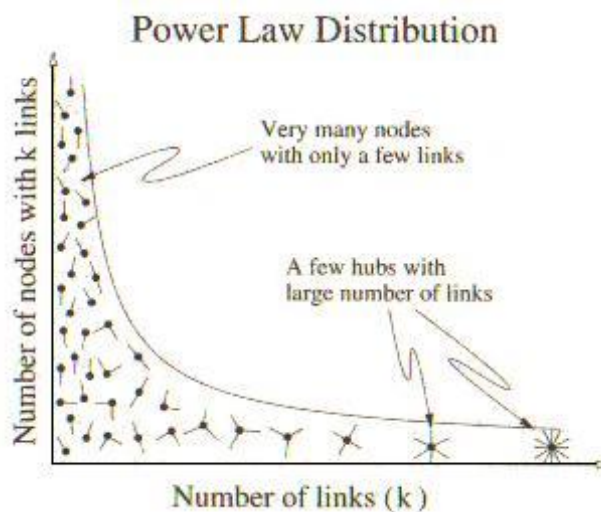


Quelques graphes particuliers

- Un graphe est **complet** si pour toute paire de sommet, il existe au moins une arête les reliant.
- Graphe **biparti** : partition des sommets en 2 ensembles U et V telle que les arêtes ont leurs extrémités dans chacun des ensembles.



- **Arbre**, for
- **Planaire** :
ne croisent.
- **Libre échelle**
de ses deg
- **Petit monde**



son coefficient d'agglomération est élevée et la distance moyenne entre deux sommets faible.

BFS (graphe **G**, sommet **s**) :

f = *CreerFile*()

Marquer(**s**)

Enfiler(**f**, **s**)

dist[**s**]=0

tant que non *FileVide*(**f**) **faire**

x = *Défiler*(**f**)

pour chaque successeur de **x** **faire**

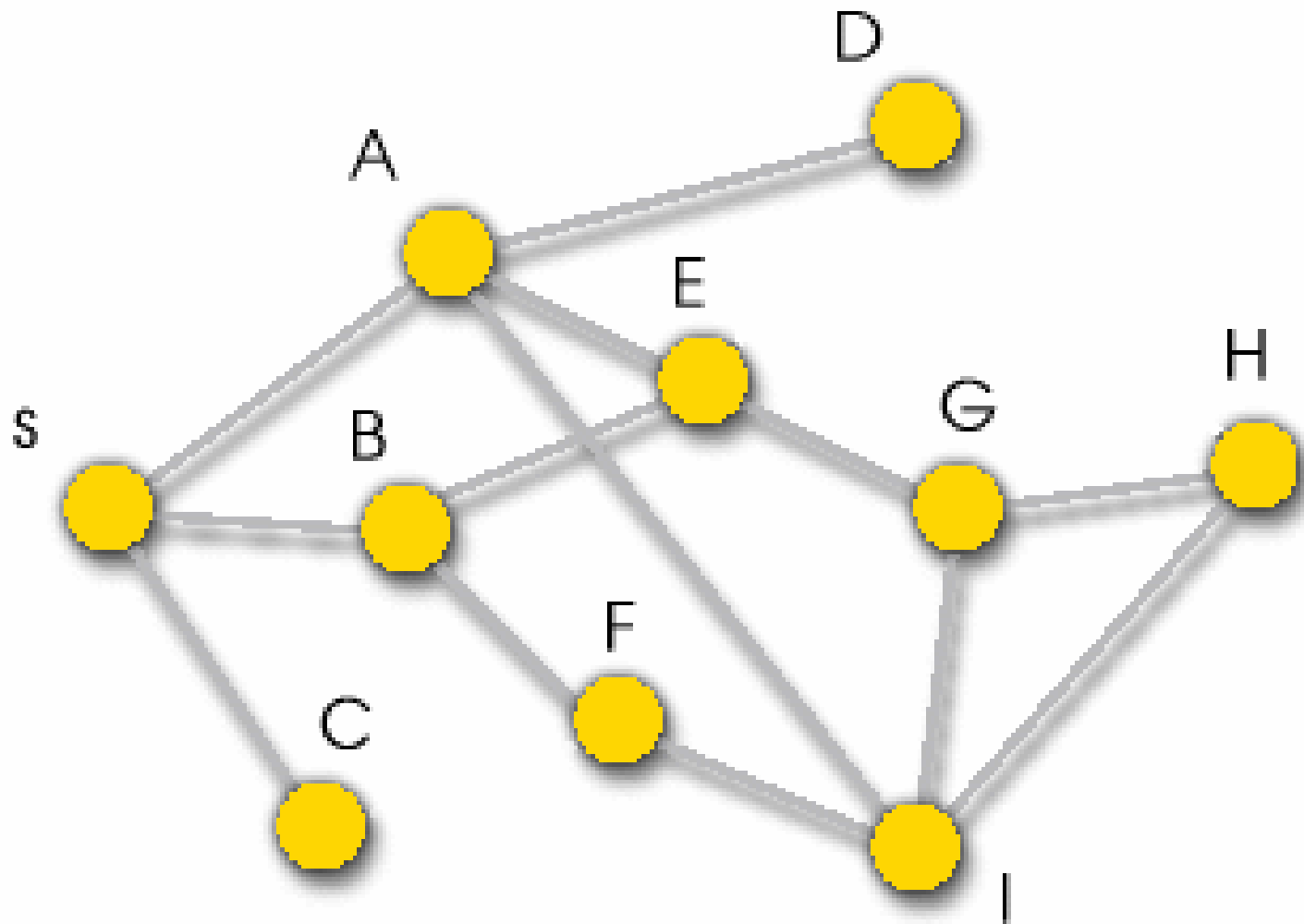
si *NonMarqué*(**successeur**) **alors**

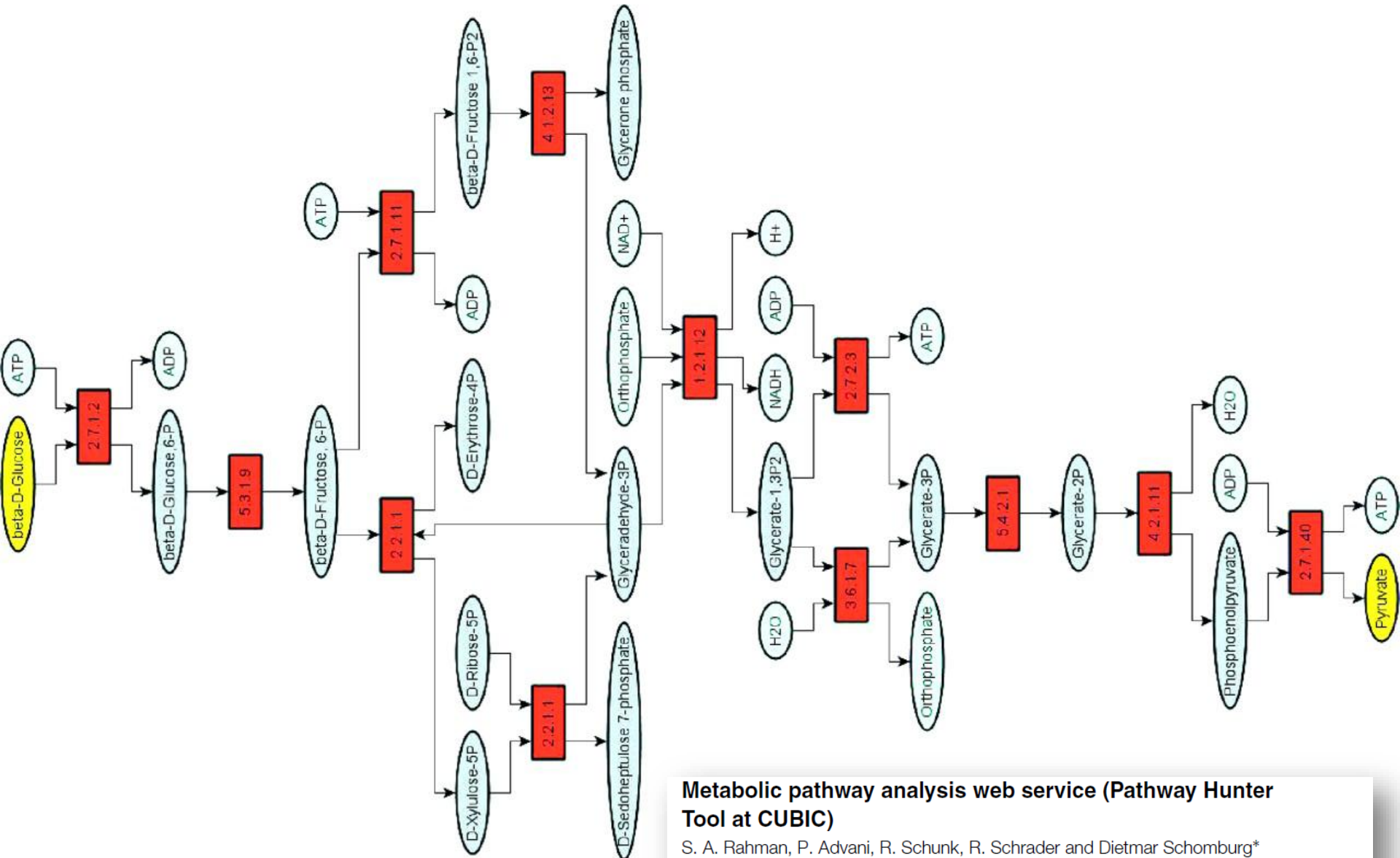
Marquer(**successeur**)

Enfiler(**f**, **successeur**)

dist[**successeur**] = **dist**[**x**] + 1

Exemple





Metabolic pathway analysis web service (Pathway Hunter Tool at CUBIC)

S. A. Rahman, P. Advani, R. Schunk, R. Schrader and Dietmar Schomburg*
 Cologne University Bioinformatics Center (CUBIC) and Institute of Biochemistry, Zùlpicher Strasse 47,
 50674 Kùln, Germany

Received on August 12, 2004; revised on October 21, 2004; accepted on October 21, 2004

```

BFS( $G, s$ )
1  for each vertex  $u \in V[G] - \{s\}$ 
2      do  $color[u] \leftarrow WHITE$ 
3           $d[u] \leftarrow \infty$ 
4           $\pi[u] \leftarrow NIL$ 
5   $color[s] \leftarrow GRAY$ 
6   $d[s] \leftarrow 0$ 
7   $\pi[s] \leftarrow NIL$ 
8   $Q \leftarrow \emptyset$ 
9  ENQUEUE( $Q, s$ )
10 while  $Q \neq \emptyset$ 
11     do  $u \leftarrow DEQUEUE(Q)$ 
12         for each  $v \in Adj[u]$ 
13             do if  $color[v] = WHITE$ 
14                 then  $color[v] \leftarrow GRAY$ 
15                      $d[v] \leftarrow d[u] + 1$ 
16                      $\pi[v] \leftarrow u$ 
17                     ENQUEUE( $Q, v$ )
18      $color[u] \leftarrow BLACK$ 

```

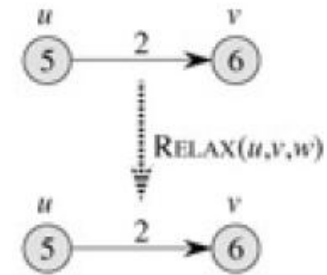
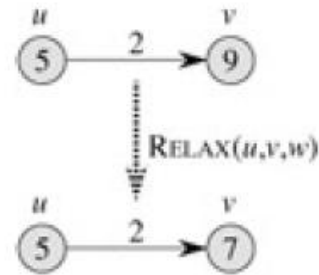

INITIALIZE-SINGLE-SOURCE(G, s)

1 **for** each vertex $v \in V[G]$

2 **do** $d[v] \leftarrow \infty$

3 $\pi[v] \leftarrow \text{NIL}$

4 $d[s] \leftarrow 0$



RELAX(u, v, w)

1 **if** $d[v] > d[u] + w(u, v)$

2 **then** $d[v] \leftarrow d[u] + w(u, v)$

3 $\pi[v] \leftarrow u$

BELLMAN-FORD(G, w, s)

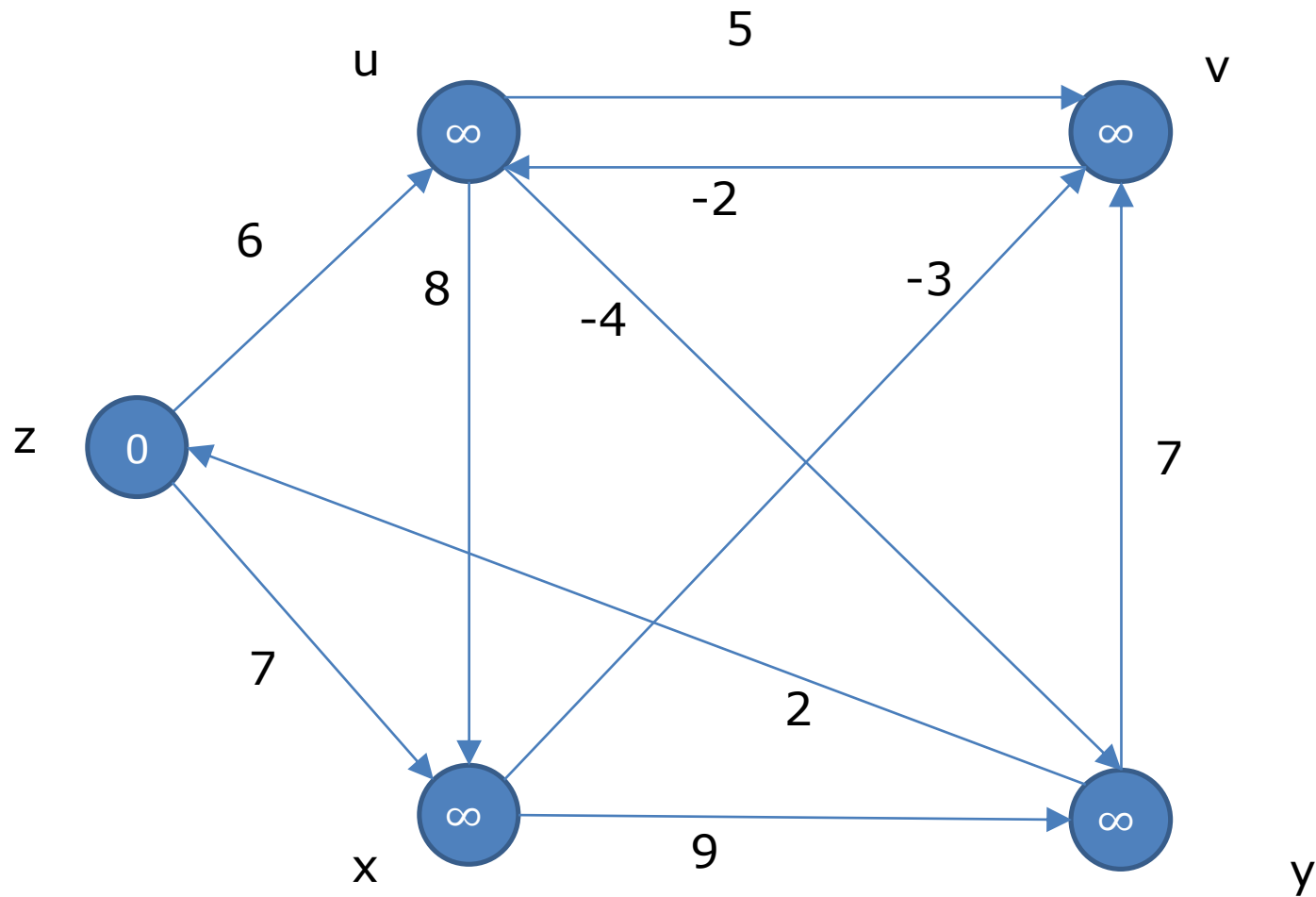
1 INITIALIZE-SINGLE-SOURCE(G, s)

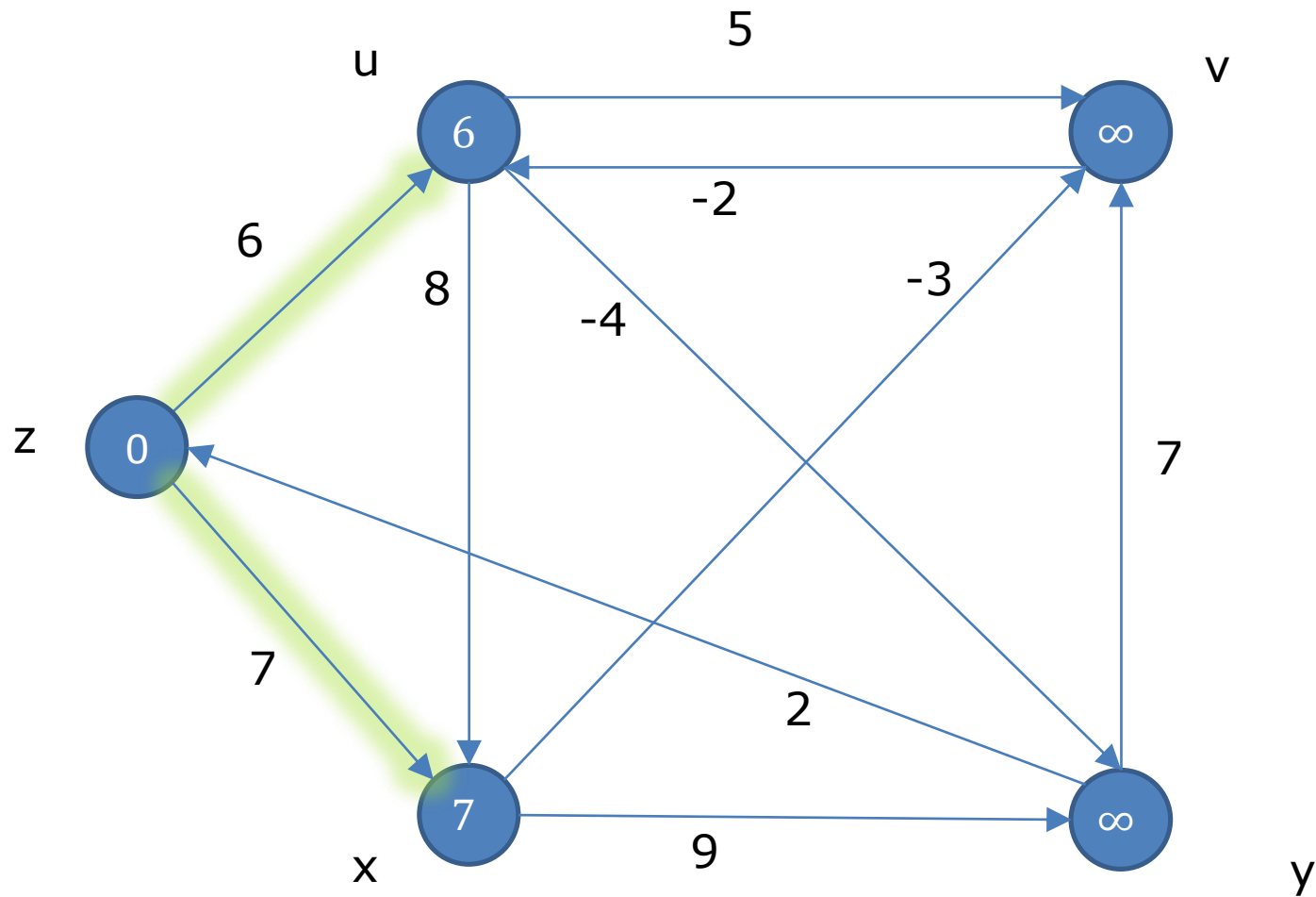
2 **for** $i \leftarrow 1$ **to** $|V[G]| - 1$

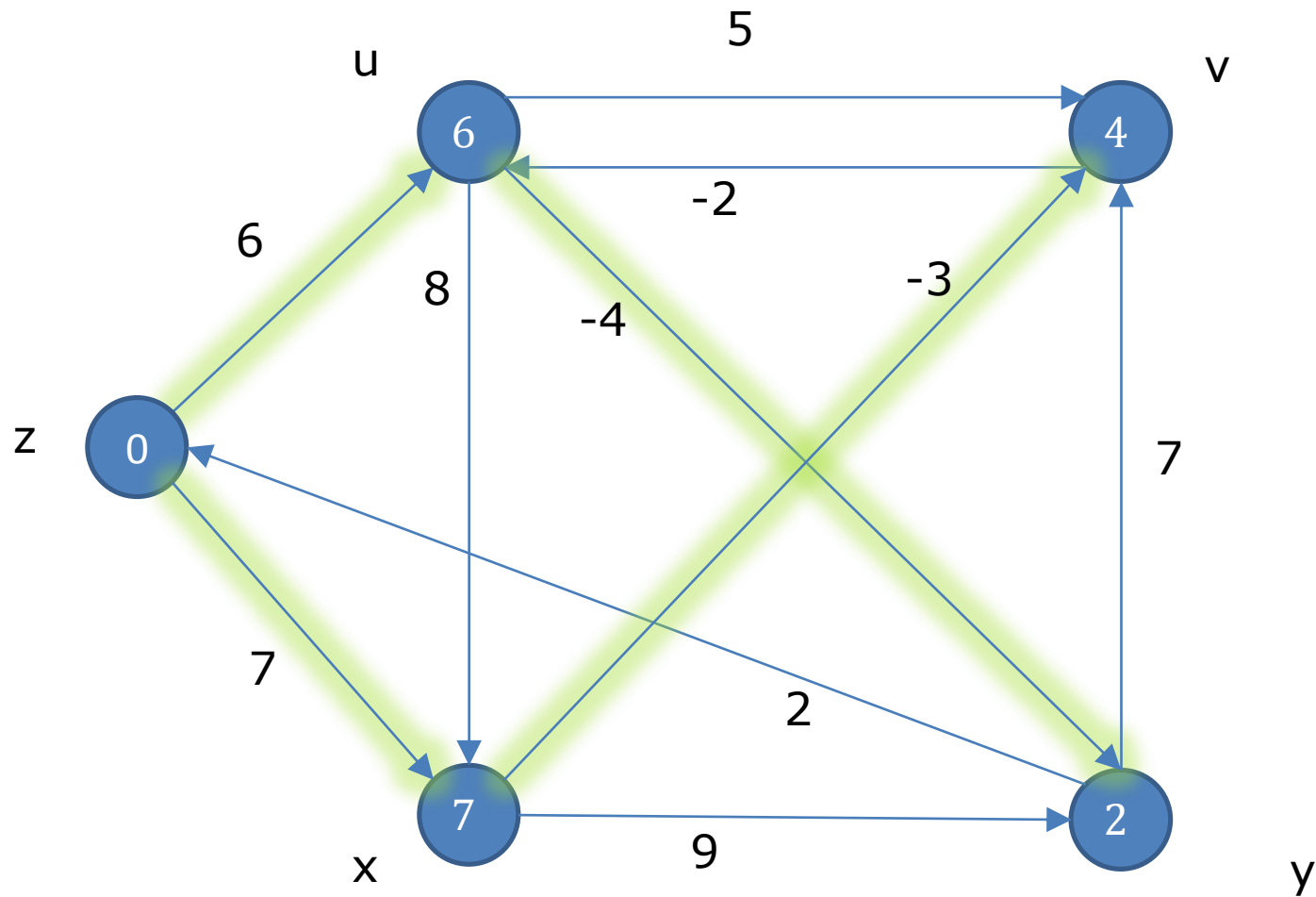
3 **do for** each edge $(u, v) \in E[G]$

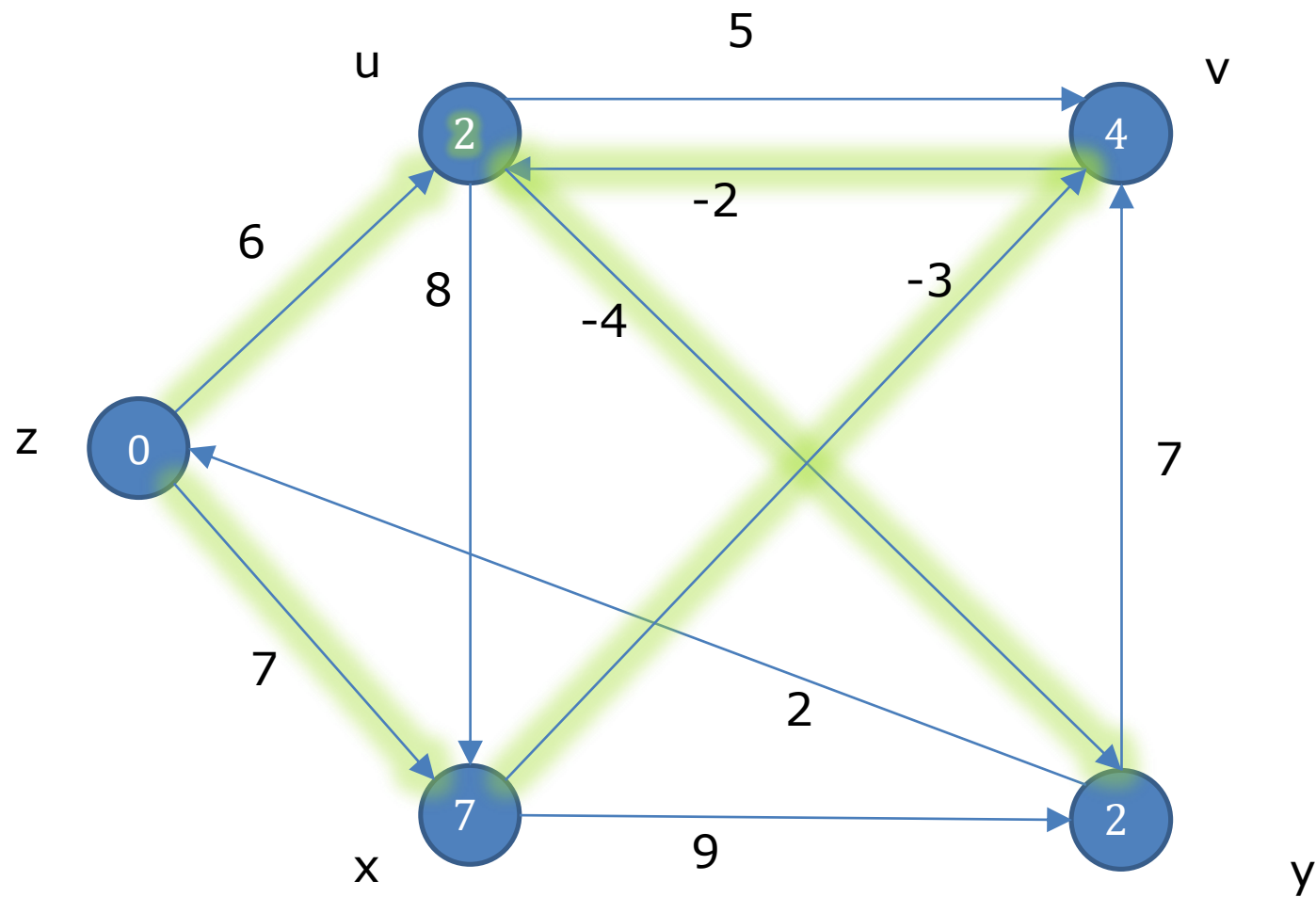
4 **do** RELAX(u, v, w)

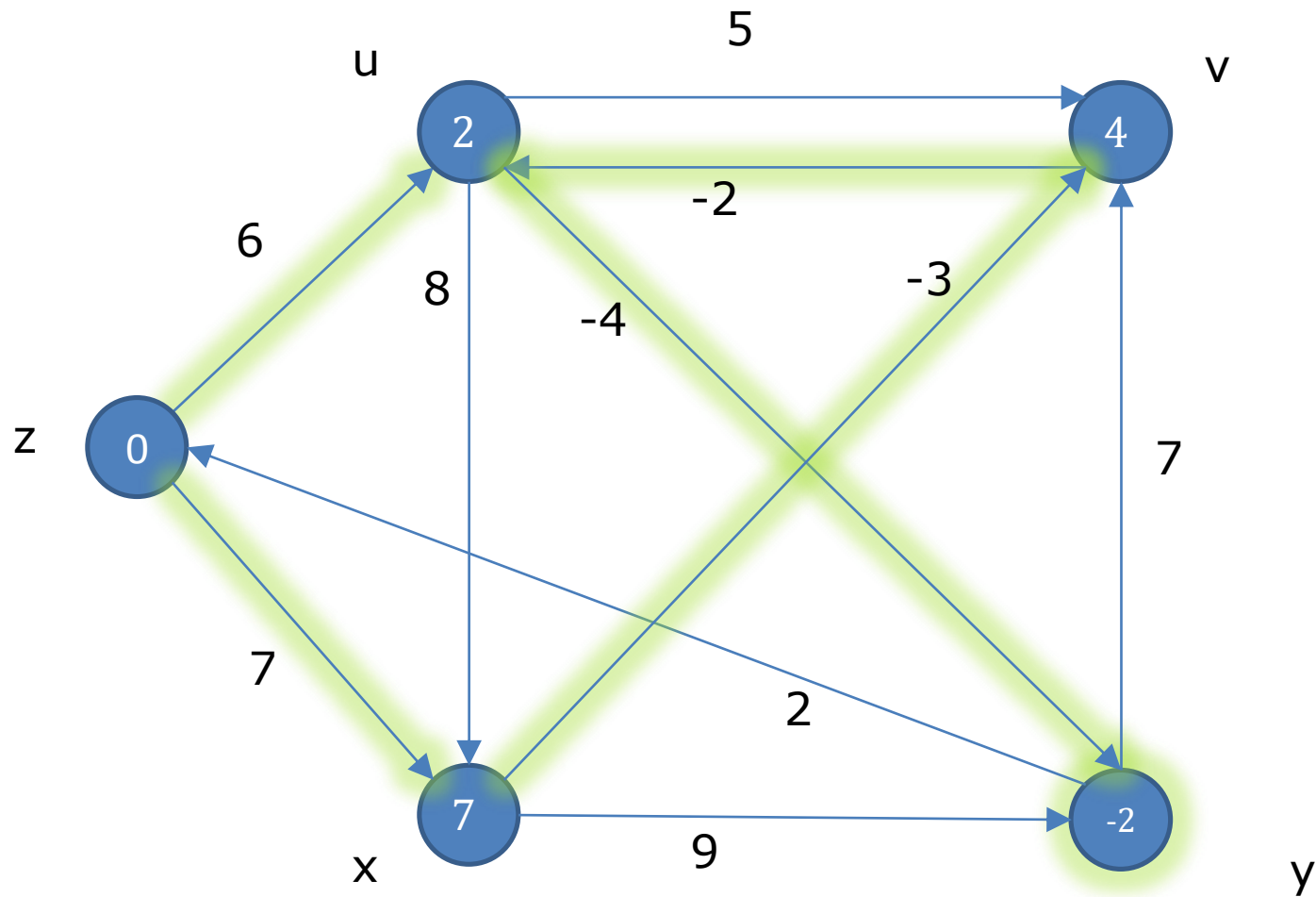
Exemple



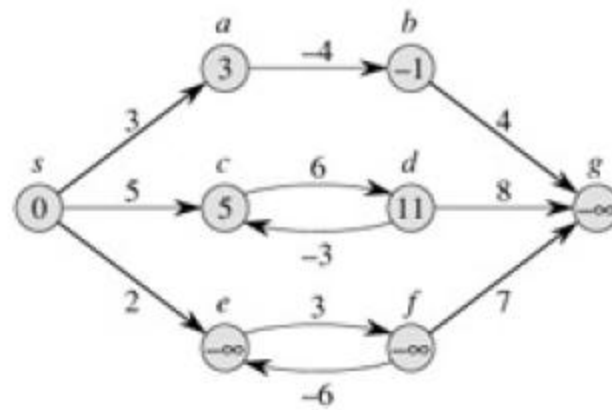
Exemple : 1^{ère} itération

Exemple : 2^{ème} itération

Exemple : 3^{ème} itération

Exemple : 4^{ème} itération

Cycle de longueurs négatives



- Sommet source s et matrice d'adjacence W
- Initialisation
 - ♦ un tableau D contenant $D[x]$ la plus courte distance de s à x . Au départ, $D[x] = \text{infini}$ pour tout x (excepté $D[s] = 0$)
 - ♦ un tableau P contenant $P[x]$ le prédécesseur de x dans le chemin de plus courte distance allant de s à x . Au départ, $P[x] = \text{rien}$ pour tout x
 - ♦ un ensemble C de sommets qu'il reste à parcourir. Au départ, $C = V$

- Boucle principale

tant que C non vide

x = *ExtraireLePlusProche*(**C**)

pour chaque successeur de **x** faire

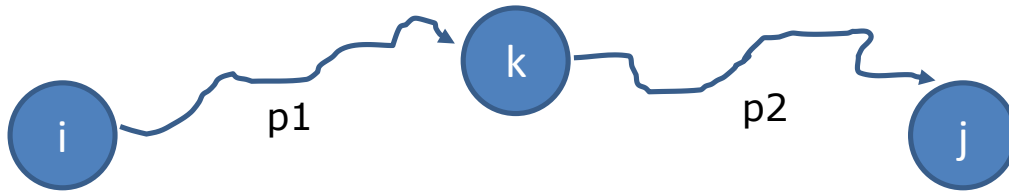
si $D[x] + W[x, \text{successeur}] < D[\text{successeur}]$ alors

$D[\text{successeur}] = D[x] + W[x, \text{successeur}]$

$P[\text{successeur}] = x$

Algorithme de Floyd-Warshall

Structure d'un plus court chemin :



On note

$p = \{v_1, v_2, \dots, v_l\}$ un chemin simple

$V = \{1, 2, \dots, n\}$ avec n l'ordre du graphe

$\{1, 2, \dots, k\}$ un sous-ensemble de V

Si k est un sommet intermédiaire

→ p_1 est un plus court chemin de i à k

→ p_2 est un plus court chemin de k à j

Formulation récursive pour programmation dynamique :

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{si } k = 0 \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{si } k \geq 1 \end{cases}$$

Longueurs des plus courts chemins entre toutes les paires de sommets

- Algorithme de Floyd-Warshall
- A partir de la matrice d'adjacence W

$D = W$ # D : longueur du plus court chemin entre paires de sommets

$N[i,j] = j$ si $(i,j) \in E$, **NIL** sinon

```

pour k de 1 à n           # sommet intermédiaire
  pour i de 1 à n         # sommet source
    pour j de 1 à n       # sommet destination
      si  $D[i,k]+D[k,j] < D[i,j]$  alors
         $D[i,j] = D[i,k]+D[k,j]$ 
         $N[i,j] = N[i,k]$  # successeur de i pour aller à j

```

- Remarque : La dernière ligne n'est nécessaire que si l'on souhaite reconstruire le plus court chemin entre 2 sommets
- Calcul du plus court chemin entre i et j à partir de D et N

si $D[i,j]$ est infinie alors

il n'y a pas de chemin entre i et j

chemin = InitialiserChemin(**i**)

k = $N[i,j]$

tant que **k** est défini faire

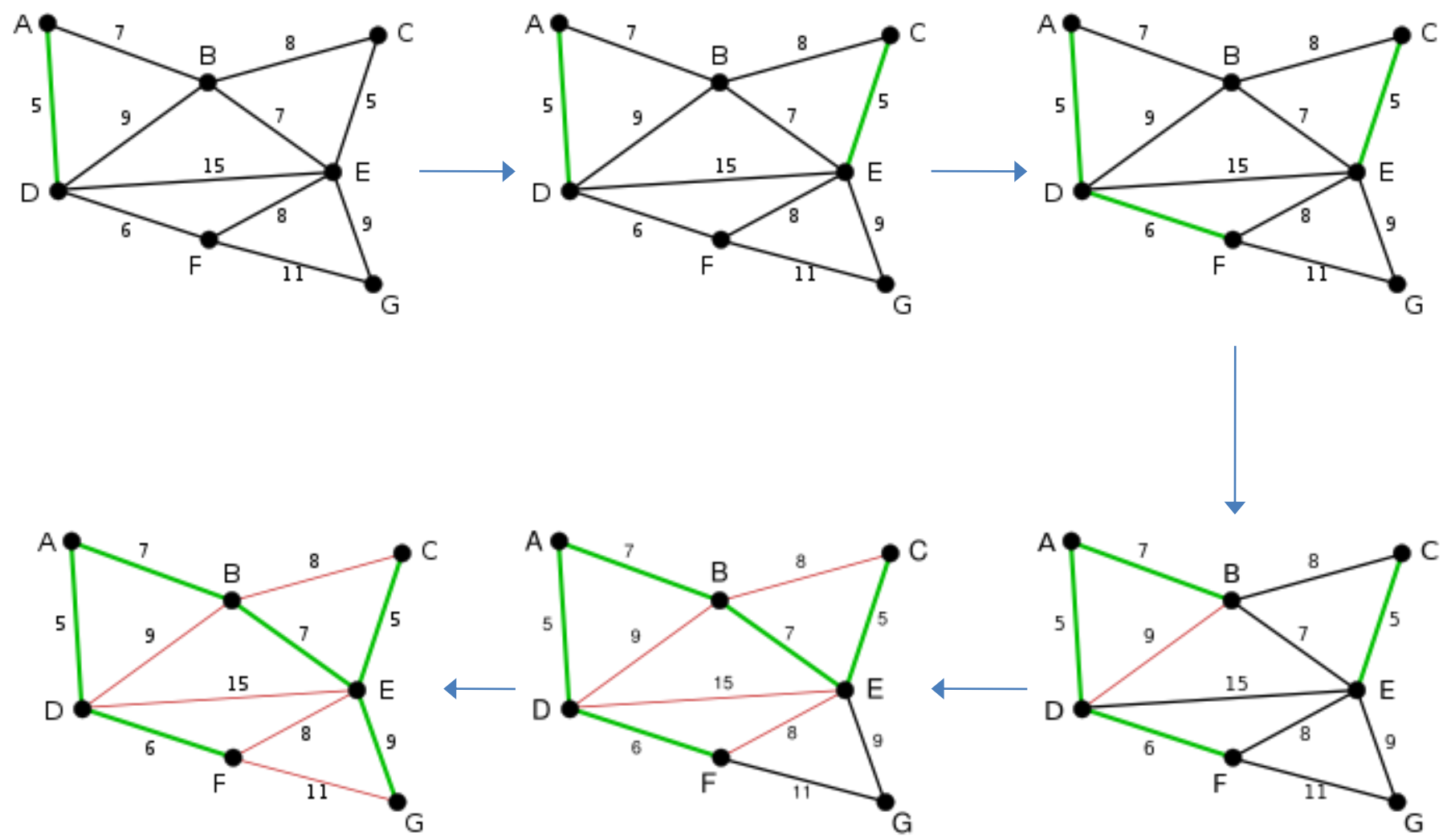
Ajouter(**chemin**, **k**)

k = $N[k,j]$

Ajouter(**chemin**, **j**)

Afficher(**chemin**)

Arbre couvrant de poids minimum : illustration de l'algorithme de Kruskal

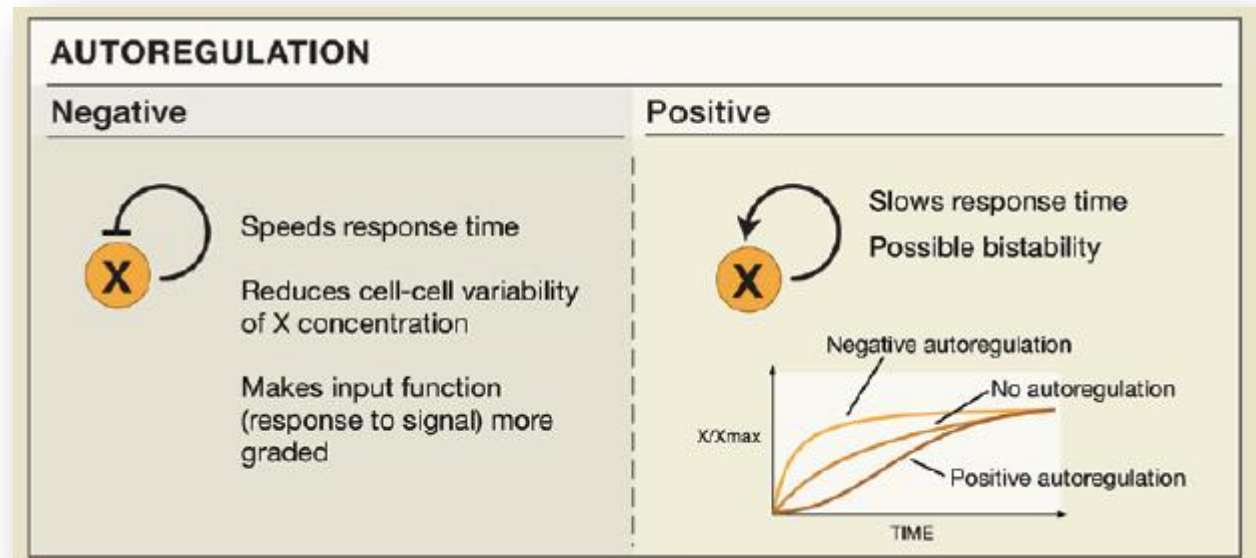


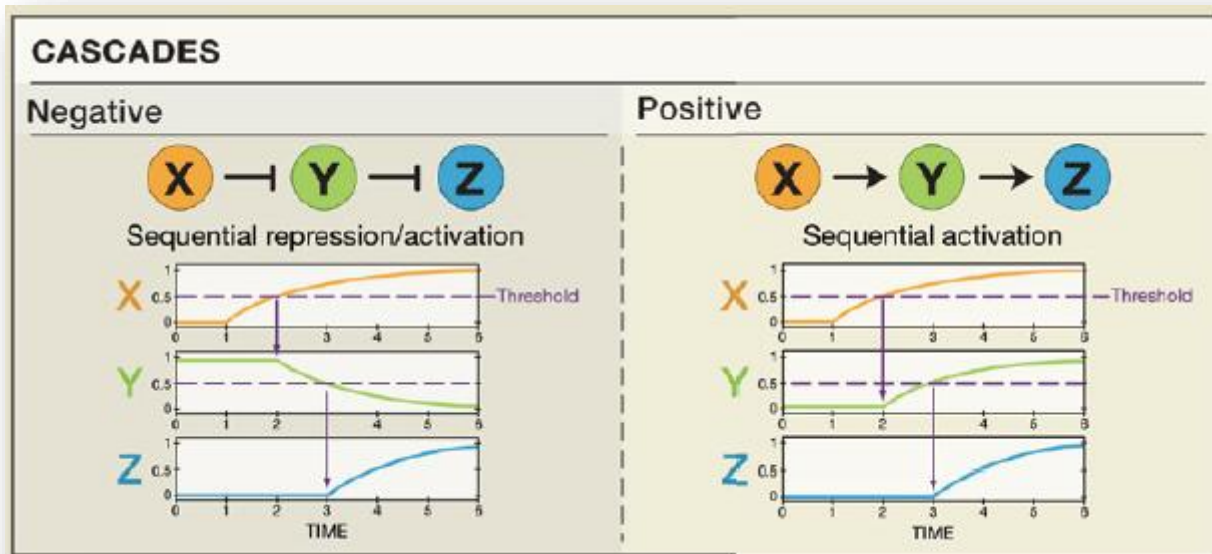
Un exemple de matrice de distance entre ST

	ST16	ST17	ST14	ST15	ST12	ST13	ST10	ST11	ST18	ST19	ST30	ST31	ST32	ST33	ST34	ST35	ST36	ST4	ST5	ST6	ST7	ST1	ST2	
	ST3	ST8	ST9	ST29	ST28	ST23	ST22	ST21	ST20	ST27	ST26	ST25	ST24											
ST16	0	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
ST17	1	0	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
ST14	1	1	0	1	1	1	1	1	1	2	2	2	2	1	2	2	1	2	2	2	2	2	2	2
ST15	1	1	1	0	1	1	1	1	1	2	2	2	2	2	1	2	2	1	2	2	2	2	2	2
ST12	1	1	1	1	0	1	1	1	1	2	2	1	2	2	2	2	2	2	2	2	2	1	2	2
ST13	1	1	1	1	1	0	1	1	1	2	2	2	1	2	2	2	2	2	2	2	2	1	2	2
ST10	1	1	1	1	1	1	0	1	1	1	2	2	2	2	2	2	2	2	2	2	2	1	2	2
ST11	1	1	1	1	1	1	1	0	1	1	2	1	2	2	2	2	2	2	2	2	2	0	2	2
ST18	1	1	1	1	1	1	1	1	0	1	2	2	2	2	2	2	2	2	2	2	2	1	2	2
ST19	1	1	1	1	1	1	1	1	1	0	2	2	2	2	2	2	2	2	2	2	2	1	2	2
ST30	2	2	2	2	2	2	1	2	2	2	0	1	1	1	1	1	1	2	2	2	2	2	1	2
ST31	2	2	2	2	2	2	2	1	2	2	1	0	1	1	1	1	1	2	2	2	2	1	2	2
ST32	2	2	2	2	1	2	2	2	2	2	2	1	1	0	1	1	1	1	2	2	2	2	1	2
ST33	2	2	2	2	2	1	2	2	2	2	2	1	1	1	0	1	1	1	2	2	2	2	0	2
ST34	2	2	1	2	2	2	2	2	2	2	2	1	1	1	1	0	1	1	1	2	2	2	2	1
ST35	2	2	2	1	2	2	2	2	2	2	2	1	1	1	1	1	0	1	2	1	2	2	2	1
ST36	1	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	0	2	2	1	2	2	2
ST4	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	1	1	1
ST5	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	1	1	1
ST6	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	1	1	1
ST7	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
ST1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
ST2	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
ST3	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
ST8	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	0	1	1	0	1	1
ST9	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1
ST29	2	2	2	2	2	2	2	2	1	2	2	2	2	2	2	2	2	2	2	2	2	1	2	2
ST28	2	2	2	2	2	2	2	1	2	2	2	2	2	2	2	2	2	2	2	2	2	1	2	2
ST23	2	2	2	2	2	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	2	2
ST22	2	2	2	2	1	2	2	2	2	2	2	2	2	2	2	2	2	2	0	2	2	2	1	1
ST21	2	2	2	2	2	2	1	2	2	2	1	2	2	2	2	2	2	2	2	2	2	1	1	1
ST20	2	2	2	2	2	1	2	2	2	1	2	2	2	2	2	2	2	2	2	2	2	1	2	2
ST27	2	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	2	1	2	2	2
ST26	1	2	2	2	2	2	2	2	2	2	2	2	2	2	1	2	2	1	2	2	2	1	1	1
ST25	2	2	2	1	2	2	2	2	2	2	2	2	2	2	1	2	2	1	2	2	2	1	1	1
ST24	2	2	1	2	2	2	2	2	2	2	2	2	1	2	2	1	2	2	2	1	1	1	1	1

- Motif : Sous-graphe retrouvé de manière récurrente dans des réseaux biologiques dont la fréquence d'apparition est supérieure à son nombre d'occurrences dans un graphe aléatoire.
- Idée : architecture modulaire des réseaux biologiques
 - ♦ chaque motif possède une dynamique spécifique en terme de modulation ou d'intégration d'un signal et donc une dynamique fonctionnelle spécifique.

- Négative
 - ◆ ex: un facteur de transcription réprime la transcription du gène le codant.
- Positive
 - ◆ peut engendrer des distributions bimodales de type tout ou rien.

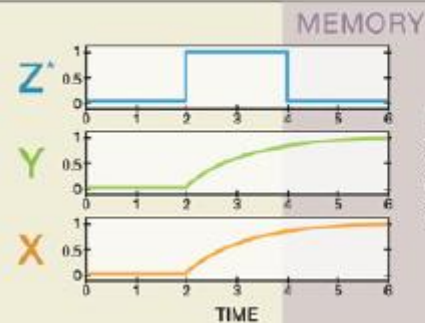
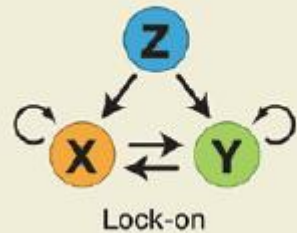




Positive-Feedback loops

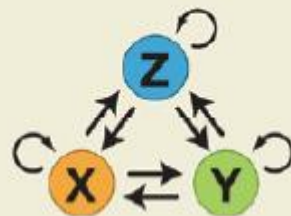


Regulated double-positive



Memory:
X, Y stay ON
after input Z
turns OFF

Fully connected triad

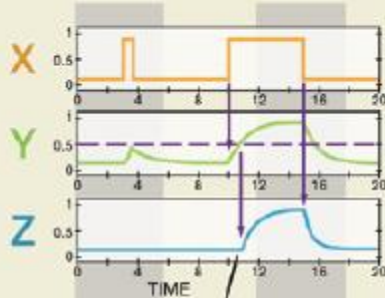
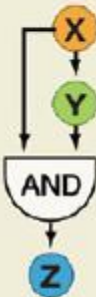


Feedforward loops (FFLs)

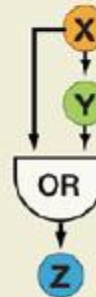
- Très fréquemment observés

Coherent type I

AND gate

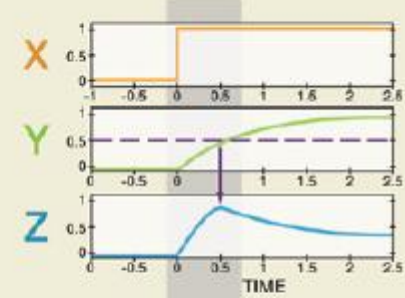


OR gate



Filters transient

Incoherent type I



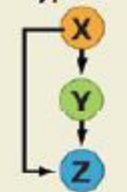
Pulse generator

Filters transient
ON
input signals

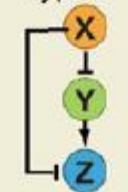
Delayed
ON
response

The 8 FFL types

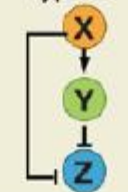
Coherent type 1



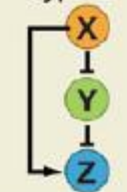
Coherent type 2



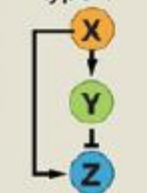
Coherent type 3



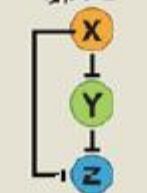
Coherent type 4



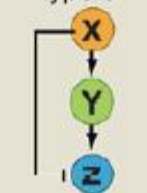
Incoherent type 1



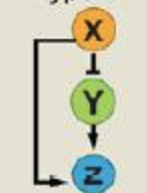
Incoherent type 2



Incoherent type 3



Incoherent type 4



Coherent type 1 and Incoherent type 1
are the most common in transcription networks

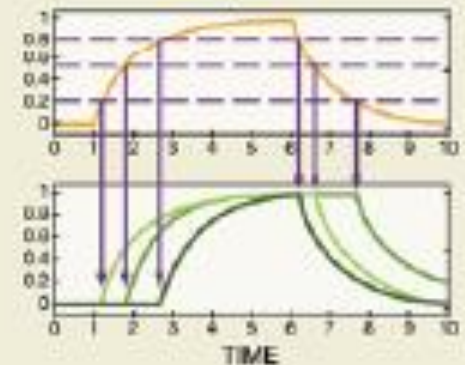
at

Pulse
generator

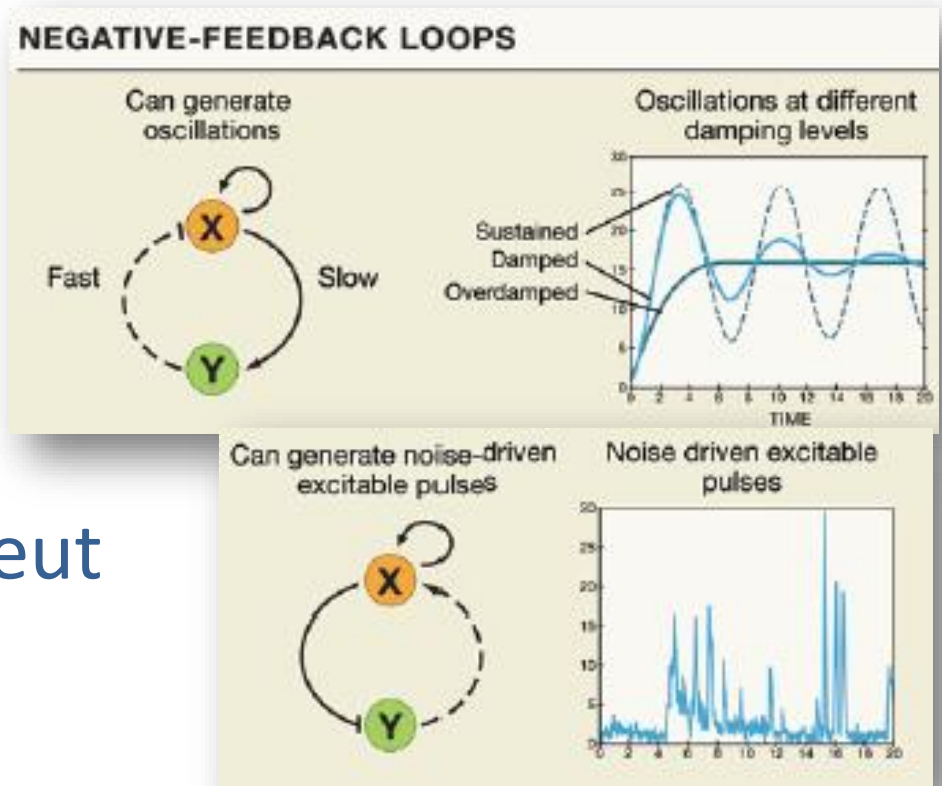
- X régule un ensemble de gènes cibles dont typiquement lui-même.
- permet l'expression coordonnée de gènes impliqués dans un même processus
- peut générer une expression temporelle à travers différents seuils pour chacune des cibles

SINGLE-INPUT MODULE (SIM)

Can generate temporal program of expression
(e.g., just-in-time transcription)



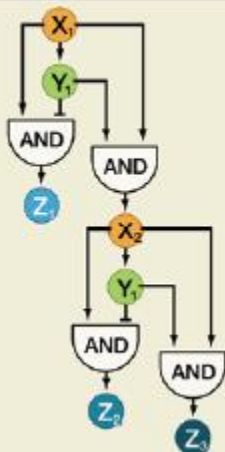
- souvent des interactions avec des échelles de temps différentes, asymétriques :
 - ♦ l'activation lente (transcription de Y) et la dégradation rapide de X peut générer des oscillations



- ♦ le schéma opposé peut générer du bruit

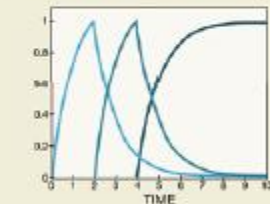
- dynamique plus élaborée
 - ◆ comportement de type file
- Intégration du signal avec entrées et sorties multiples

INTEGRATED FFLs



Example: *B. subtilis* sporulation

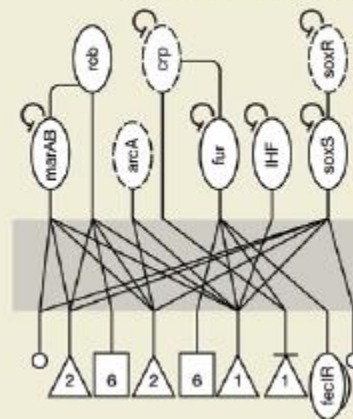
Acts as FFL for each output gene
Can generate FIFO temporal order



Integrated coherent and incoherent FFLs generate series of expression pulses

INTEGRATED MOTIFS AND DENSE OVERLAPPING REGULONS

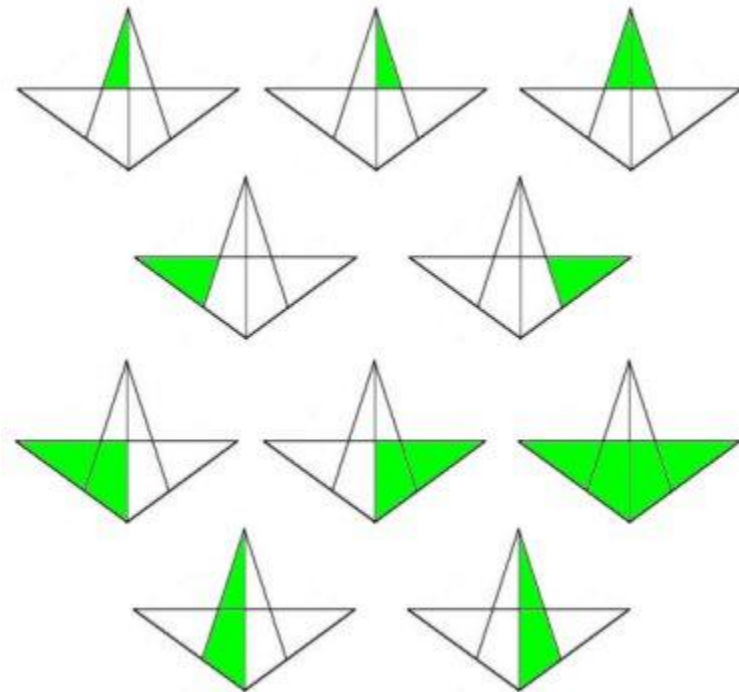
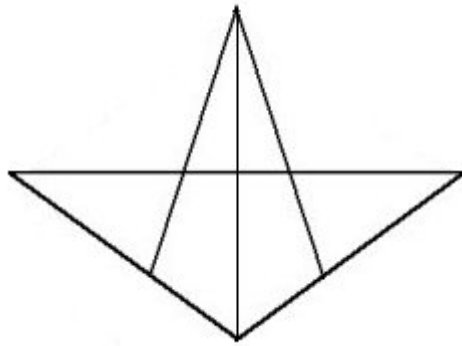
FFLs, SIMs are combined into dense overlapping regulons (DORs)



Example: Drug and superoxide DOR in *E. coli* transcription network

- Transcription factor (TF)
- Dense overlapping regulon (DOR)
- Single-input module (SIM)
- △ Coherent feedforward loop
- ▽ Incoherent feedforward loop
- Individual gene or operon

- Combien y a-t-il de triangles ?



- Problème difficile
 - ◆ taille du graphe
 - ◆ taille des motifs recherchés
- 2 approches
 - ◆ à partir du graphe, compter les motifs qui apparaissent (network centric)
 - ◆ à partir des motifs possibles, scanner le graphe pour chacun d'eux (motif centric)

- Pattern growth tree

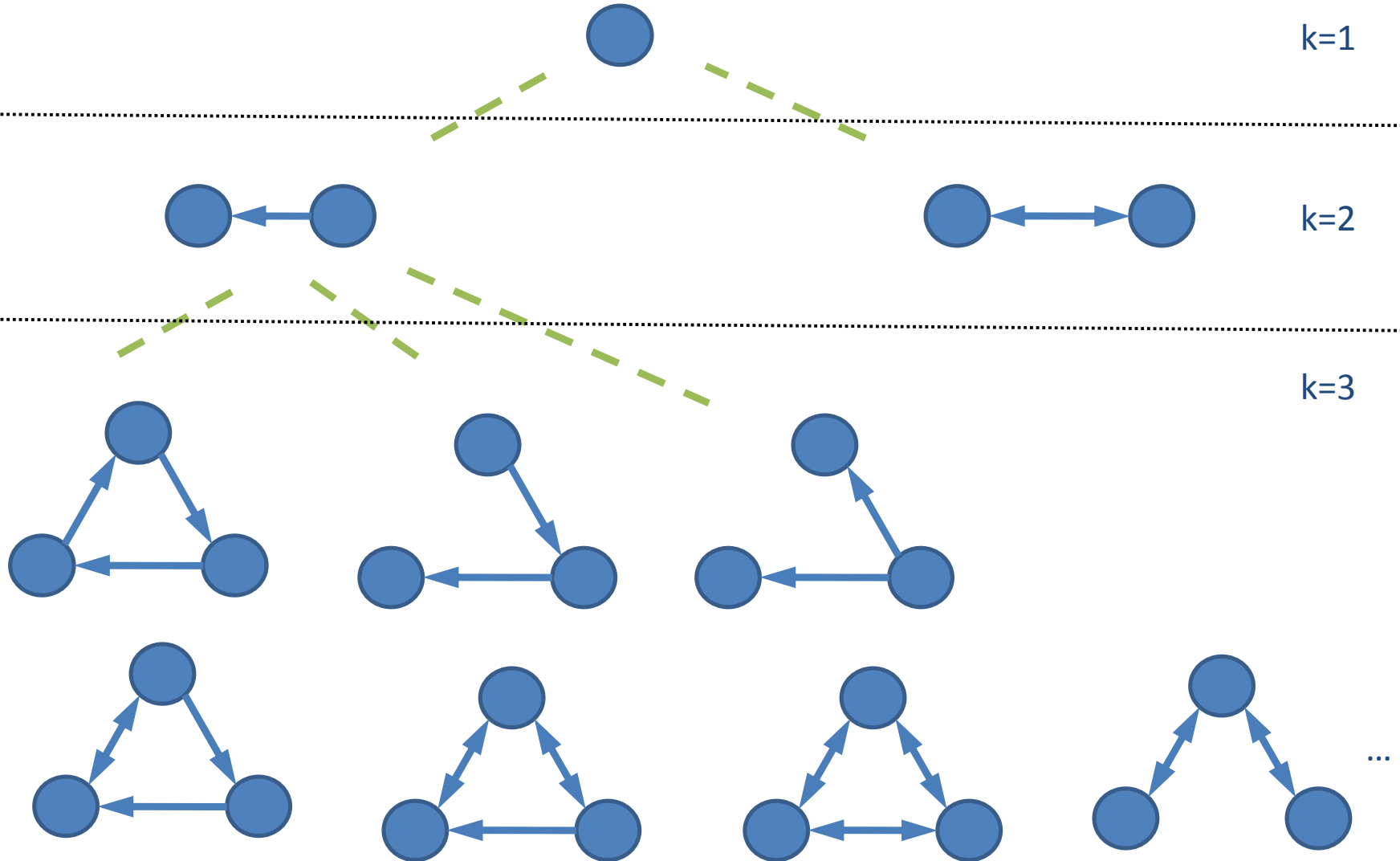


Table 8: Computational cost for Kavosh and FANMOD algorithms on Homo sapiens network (times are in seconds, rows indicate different sizes of sub-graph and columns are related to different algorithms) and the numbers of sub-graphs

	3	4	5
<i>Kavosh</i>	15	2160	29794
<i>FANMOD</i>	36	5292	-
<i>Number of sub-graphs</i>	2750397	232652426	23287189708

Kavosh: a new algorithm for finding network motifs

Zahra Razaghi Moghadam Kashani¹, Hayedeh Ahrabian^{*2,3}, Elahe Elahi⁴, Abbas Nowzari-Dalini^{2,3}, Elnaz Saberi Ansari², Sahar Asadi², Shahin Mohammadi², Falk Schreiber^{5,6} and Ali Masoudi-Nejad^{*1,3}

Address: ¹Laboratory of Systems Biology and Bioinformatics, Institute of Biochemistry and Biophysics, University of Tehran, Tehran, Iran, ²School of Mathematics and Computer Science, University of Tehran, Tehran, Iran, ³Center of Excellence in Biomathematics, University of Tehran, Tehran, Iran, ⁴School of Biology, University of Tehran, Tehran, Iran, ⁵Institute for Computer Science, Martin-Luther-University Halle-Wittenberg, Halle, Germany and ⁶Leibniz Institute of Plant Genetics and Crop Plant Research (IPK), Gatersleben, Germany

Email: Zahra Razaghi Moghadam Kashani - razaghi@ibb.ut.ac.ir; Hayedeh Ahrabian* - ahrabian@ut.ac.ir; Elahe Elahi - elahe.elahi@khayam.ut.ac.ir; Abbas Nowzari-Dalini - nowzari@ut.ac.ir; Elnaz Saberi Ansari - elnaz_ansari@khayam.ut.ac.ir; Sahar Asadi - saharasadi@gmail.com; Shahin Mohammadi - shahin.mohammadi@gmail.com; Falk Schreiber - schreiber@ipk-gatersleben.de; Ali Masoudi-Nejad* - amasoudin@ibb.ut.ac.ir

* Corresponding authors

Published: 4 October 2009

BMC Bioinformatics 2009, 10:318 doi:10.1186/1471-2105-10-318

This article is available from: <http://www.biomedcentral.com/1471-2105/10/318>

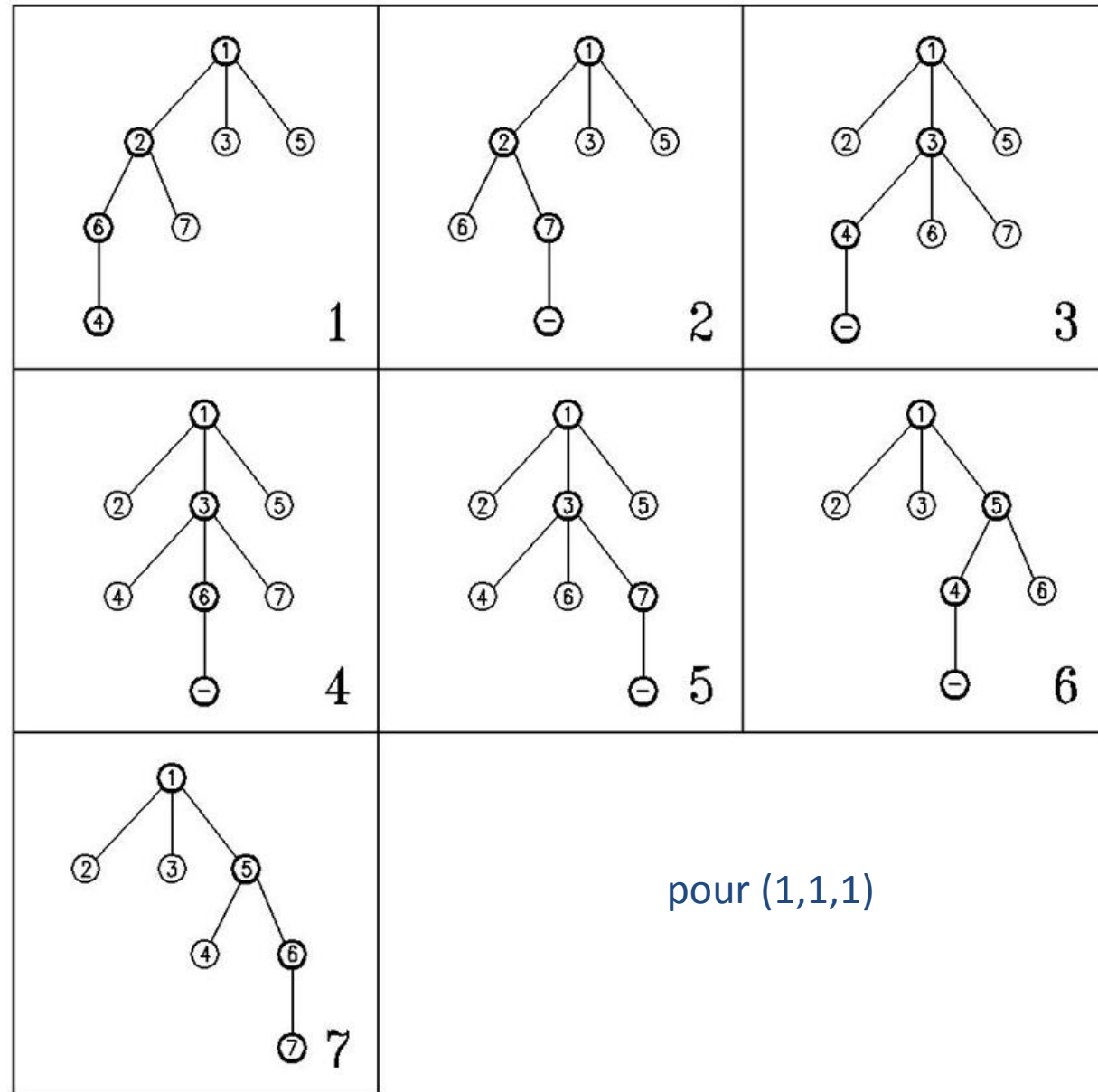
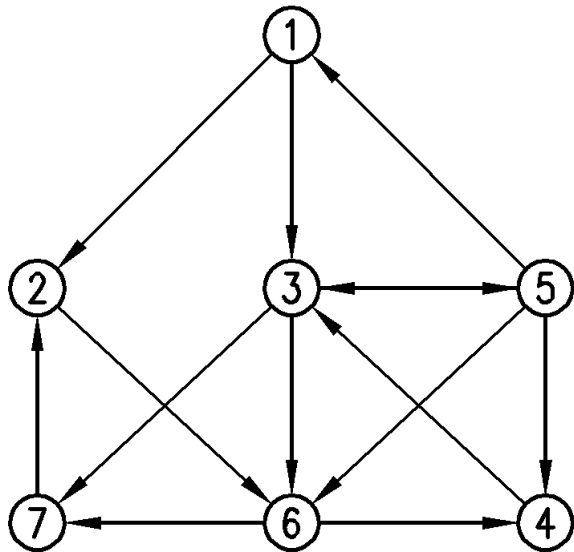
Received: 17 March 2009

Accepted: 4 October 2009

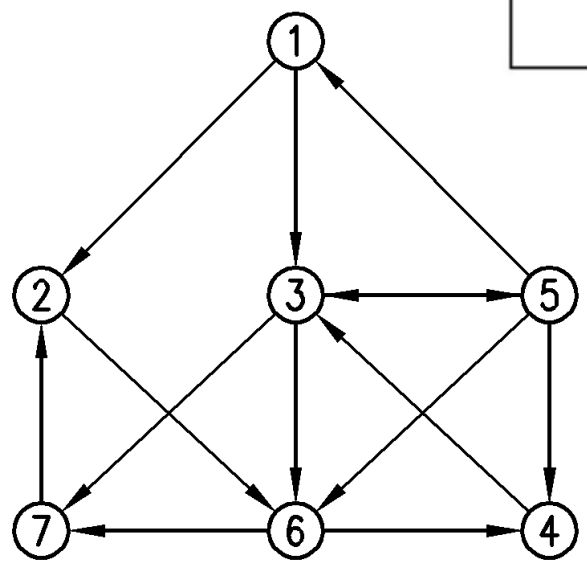
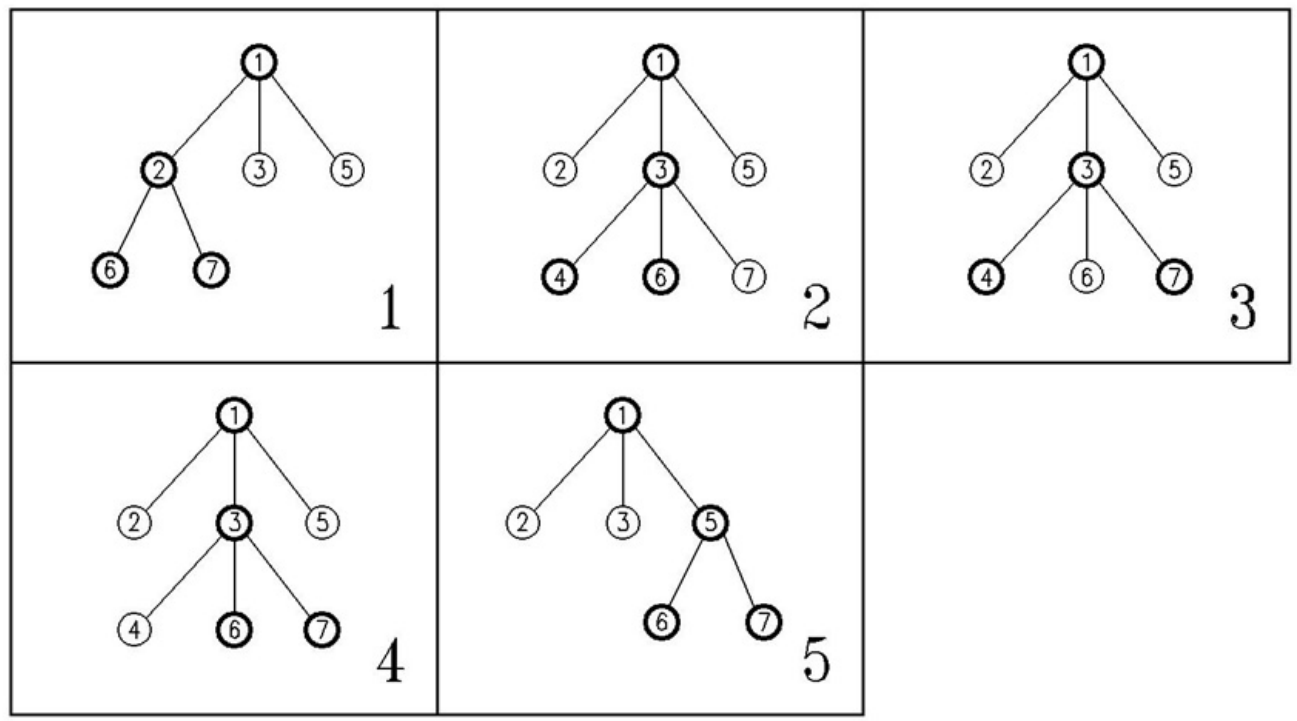
- 2 Etapes :
 - ◆ **Enumération** des sous-graphes de taille k observés contenant un sommet donné (approche network centric)
 - ◆ Elimination de ce sommet et passage au suivant (permet de générer les autres sous-graphes ne contenant pas ce sommet)
 - ◆ **Classification** des sous-graphes obtenus en classes d'isomorphisme (sous-graphes ayant même structure/topologie)

- Exploite le nombre de façons d'obtenir $k-1$ en sommant des entiers
 - ◆ ex: pour $k=4$, on a $1+1+1$ ou $1+2$ ou $2+1$ ou 3
- Utilisé pour extraire les sous-graphes à partir d'un sommet
 - ◆ Les sommets du graphe sont numérotés arbitrairement de 1 à l'ordre du graphe
 - ◆ on part du sommet de départ et on sélectionne les voisins successivement. Par exemple pour $1+1+1$, on prend 1 voisin, puis un voisin du voisin, puis un voisin du voisin du voisin (3 niveaux)
 - ◆ un sommet est marqué si il est adjacent d'un sommet sélectionné à un niveau inférieur

Enumeration des sous-graphes de taille k contenant un sommet

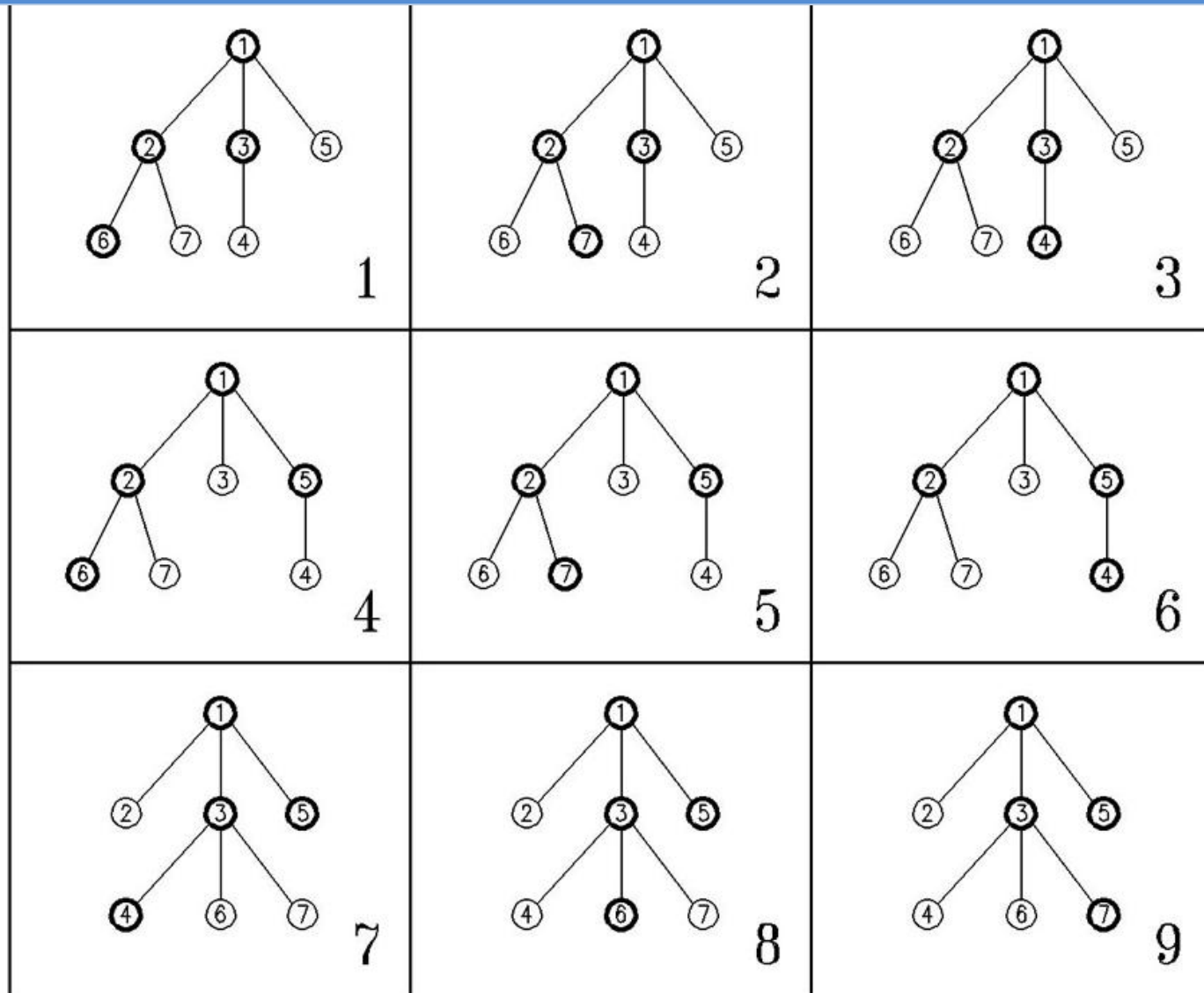
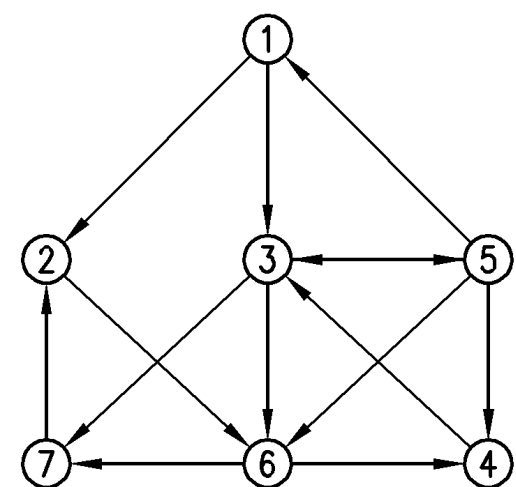


Enumeration des sous-graphes de taille k contenant un sommet



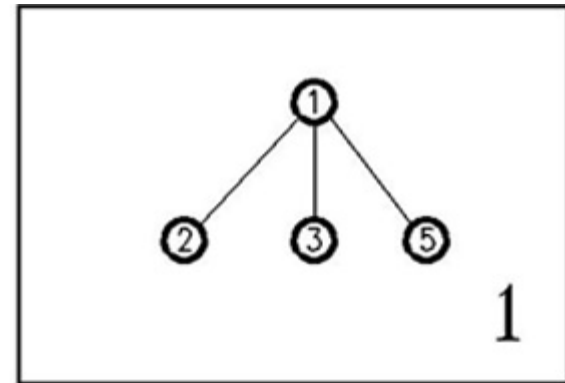
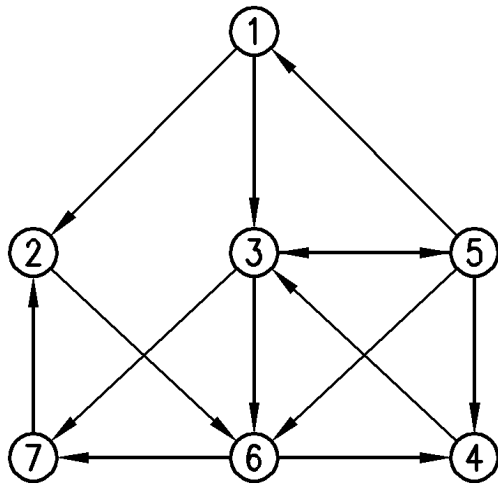
pour (1,2)

Enumeration des sous-graphes de taille k contenant un sommet



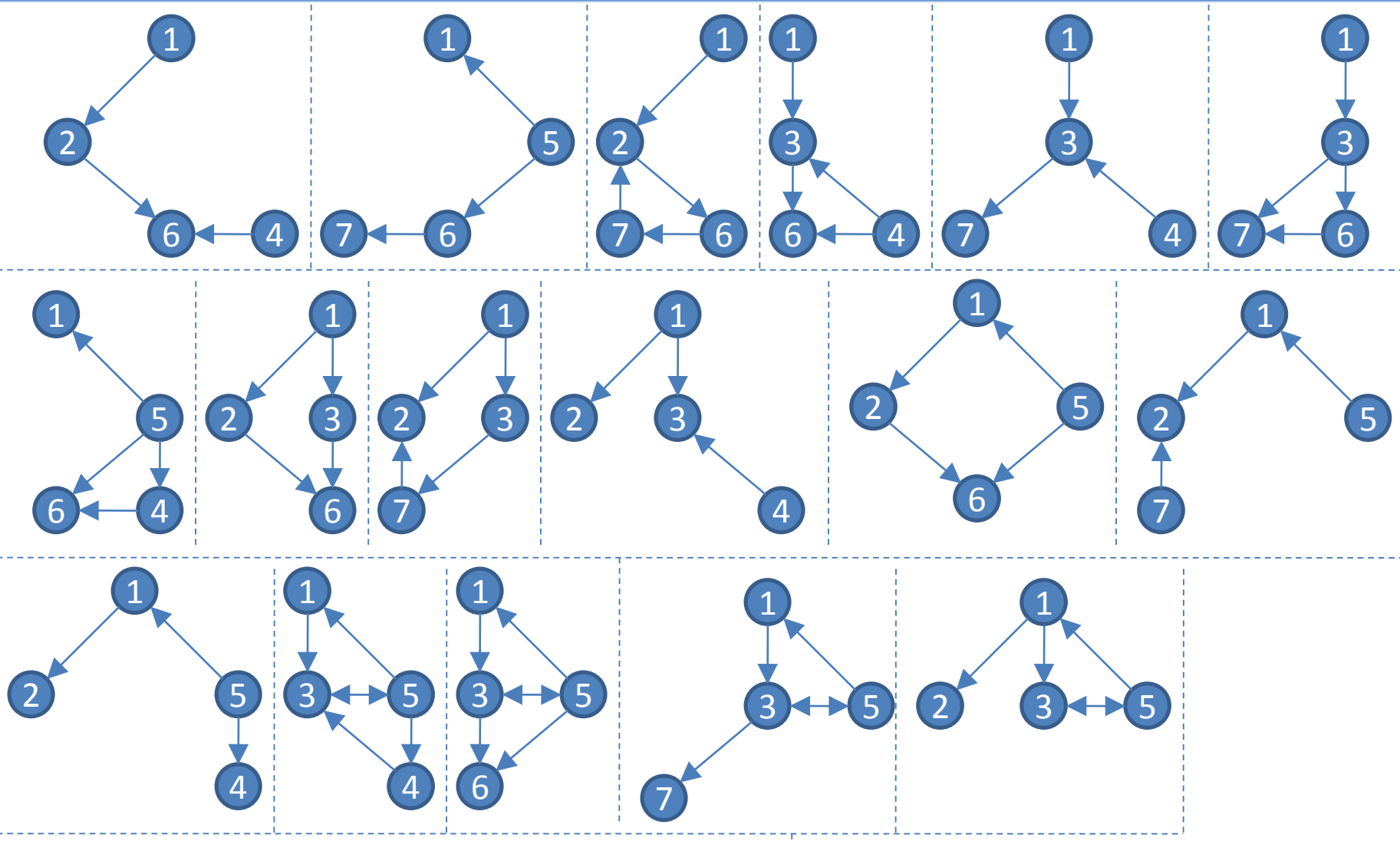
pour (2,1)

Enumeration des sous-graphes de taille k contenant un sommet

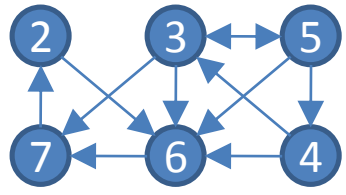


pour (3)

Sous-graphes contenant le sommet 1



Même méthode sur le graphe ci-contre (arbre de racine 2).
 Puis, en retirant le sommet de 2, ...

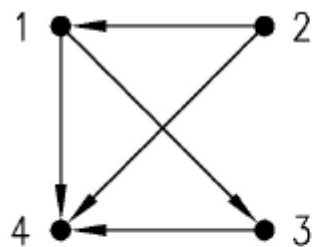


Ensuite, détermination du nombre d'occurrences de chaque motif

- 2 graphes ont même topologie si ils sont isomorphes
- Représentation canonique d'un graphe dont les sommets ne sont pas étiquetés
 - ◆ Programme NAUTY développé par McKay à partir de *Practical Graph Isomorphism*, *Congressus Numerantium*, 30 (1981) 45-87
 - ◆ Principe :
 - la matrice d'adjacence est représentée par une chaîne de caractères (concaténation des lignes)
 - réorganisation de la matrice (tri lexicographique) afin d'obtenir le mot le plus loin par rapport à l'ordre lexicographique

- Problème difficile :
 - ◆ n sommets
 - ◆ $n!$ ordres possibles
 - ◆ ex: $n=4$

a,b,c,d	b,a,c,d	d,a,b,c	c,a,b,d
a,b,d,c	b,a,d,c	d,a,c,b	c,a,d,b
a,c,b,d	b,c,a,d	d,b,a,c	c,b,a,d
a,c,d,b	b,c,d,a	d,b,c,a	c,b,d,a
a,d,b,c	b,d,a,c	d,c,a,b	c,d,a,b
a,d,c,b	b,d,c,a	d,c,b,a	c,d,b,a



	1	2	3	4
1	0	0	1	1
2	1	0	0	1
3	0	0	0	1
4	0	0	0	0

Figure 3

Sample graph with its adjacency matrix. A sample graph is shown in (a). As there are 4 vertices in this graph, there are $4!$ permutations on its vertices to indicate its different adjacency matrices and so different strings according to NAUTY description. The adjacency matrix in (b) reflects (1, 2, 3, 4) ordering of vertices. Among all different permutations (2, 1, 3, 4) ordering, creates the largest string which its related adjacency matrix is shown in (c) and this is the one known as *canonical labeling*.

- Comparaison du nombre d'occurrences observé dans le réseau par rapport à des graphes aléatoires
- Difficultés :
 - ◆ nombre de graphes aléatoires
 - ◆ modèle de génération des graphes aléatoires
- Mesures couramment utilisées :
 - ◆ Fréquence : désigne en fait le nombre d'occurrences du motifs
 - ◆ z-score : $(\text{nombre observé} - \text{moyenne dans les graphes aléatoire}) / \text{écart type}$
 - ◆ p-valeur : nombre de graphes aléatoires dans lequel le motif apparaît de manière plus fréquente / nombre de graphes aléatoires

Table 1: Total number of sub-graphs of different sizes in different networks (rows indicate different sizes of sub-graph and columns are related to different networks).

	3	4	5	6	7	8	9	10
<i>E. coli</i> (672, 1276)	2590	12896	80724	558080	4019781	29294103	212782828	1529707241
<i>S. cerevisiae</i> (688, 1079)	13150	183174	2508149	32883898	416284878	5184710063	61755820688	700928564818
<i>Social</i> (67, 182)	488	2183	10599	52156	254674	1224376	5764767	26429201
<i>Electronic</i> (97, 189)	1121	4316	19675	97038	495274	2572125	13512688	71614362

Table 2: Number of non-isomorphic sub-graphs in different networks (rows indicate different sizes of subgraph and columns are related to different networks).

	3	4	5	6	7	8	9	10
<i>E. coli</i>	12	83	590	3884	23587	136569	768121	4223040
<i>S. cerevisiae</i>	7	34	174	888	4809	27003	183307	1083282
<i>Social</i>	13	108	773	5062	30217	165958	854023	4161577
<i>Electronic</i>	4	13	49	199	907	4333	20692	96483

Table 1: Total number of sub-graphs of different sizes in different networks (rows indicate different sizes of sub-graph and columns are related to different networks).

	3	4	5	6	7	8	9	10
<i>E. coli</i> (672, 1276)	2590	12896	80724	558080	4019781	29294103	212782828	1529707241
<i>S. cerevisiae</i> (688, 1079)	13150	183174	2508149	32883898	416284878	5184710063	61755820688	700928564818

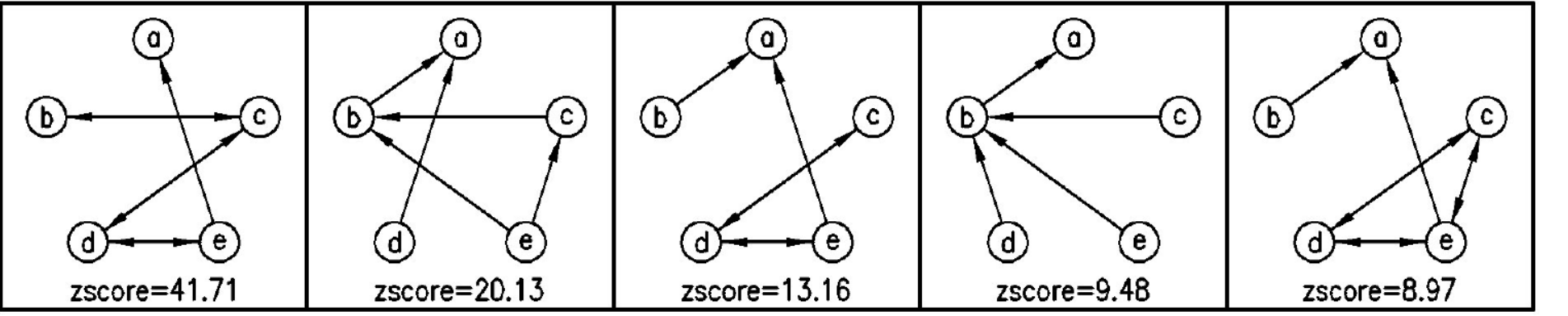
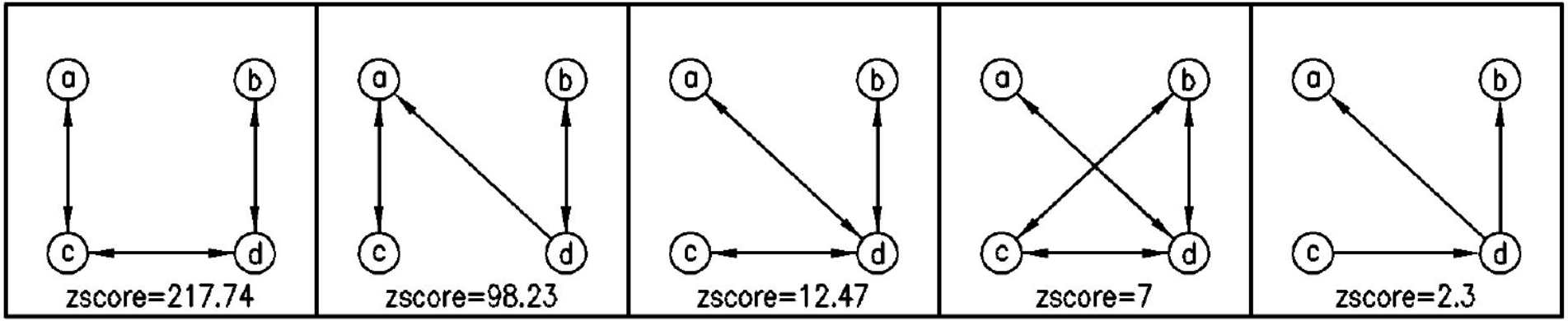
Table 3: Computational cost for different algorithms on the *E. coli* network (rows indicate different sizes of sub-graph and columns are related to different algorithms), times are in seconds.

	3	4	5	6	7	8	9	10
<i>Kavosh</i>	0.30	1.84	14.91	141.98	1374.01	13173.74	121110.31	1120560.16

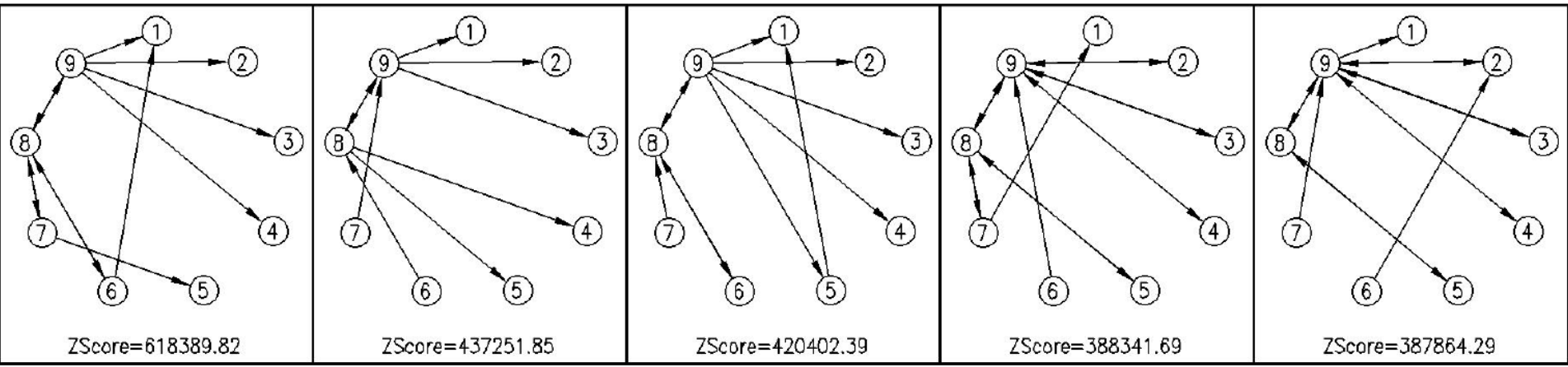
Table 4: Computational cost for different algorithms on the *S. cerevisiae* network (rows indicate different sizes of sub-graph and columns are related to different algorithms), times are in seconds.

	3	4	5	6	7	8	9	10
<i>Kavosh</i>	1.35	34.59	1003.92	20212.99	746385.86	17111178.28	337076691.32	7211199226.13

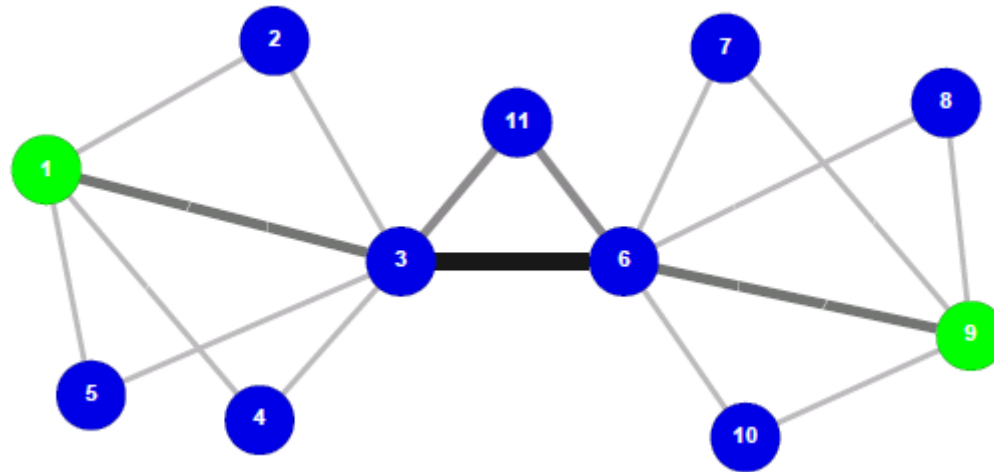
Application au réseau métabolique de *E. coli* [Kashani et al., 2009]



source : Kashani et al. 2009



- Le plus court chemin n'est pas forcément le plus pertinent



source : Dupont *et al.* 2006

BIOINFORMATICS ORIGINAL PAPER

Vol. 26 no. 9 2010, pages 1211–1218
doi:10.1093/bioinformatics/btq105

Systems biology

Advance Access publication March 12, 2010

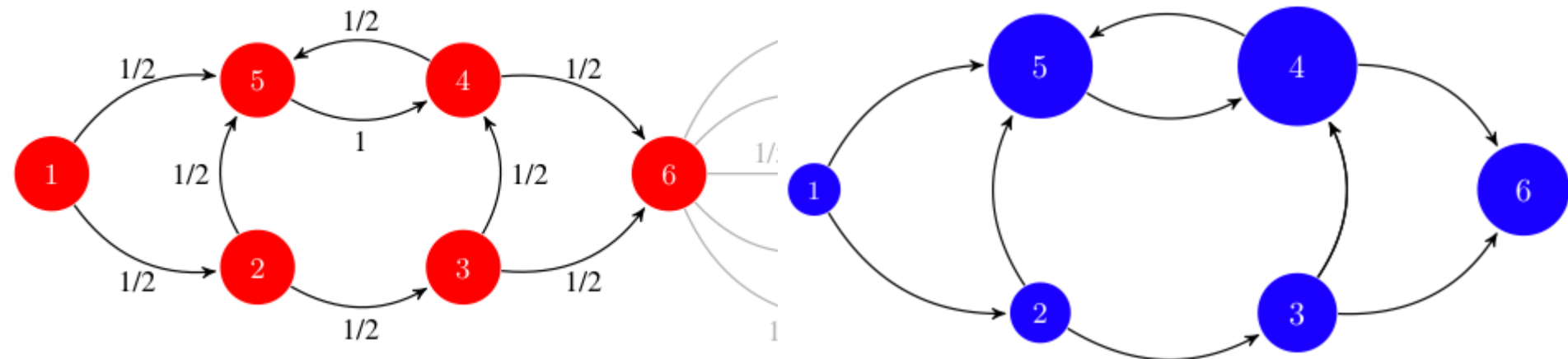
Pathway discovery in metabolic networks by subgraph extraction

Karoline Faust^{1,*}, Pierre Dupont², Jérôme Callut² and Jacques van Helden¹

¹Laboratoire de Bioinformatique des Génomes et des Réseaux (BiGRé), Université Libre de Bruxelles, Campus Plaine—CP263, Boulevard du Triomphe, 1050 Bruxelles and ²UCL Machine Learning Group, Computing Science and Engineering Department, Université catholique de Louvain, B-1348 Louvain-la-Neuve, Belgium

Associate Editor: Jonathan wren

- Principe, rang d'un nœud = proportion de temps passé sur un nœud pour une marche aléatoire infinie
 - ◆ dépend du nombre d'arcs incidents
 - ◆ du rang des prédécesseurs
- Processus Markovien



- Des approches très variées et de nombreuses méthodes existantes
- Abordées ici :
 - ◆ Graphes quelconques
 - modèle physique : Fruchterman & Reingold
 - High Dimensional Embedding
 - ◆ Arbres et arborescences
 - dessin circulaire
 - dessin non circulaire

- S'inspire de la physique des particules (et des méthodes précédentes) :
 - « *At a distance of about 1fm the strong nuclear force is attractive and about 10 times the electric force between two protons. The force decreases rapidly with increasing distance, becoming completely negligible at about 15 times this separation. When two nucleons are within about 0-4fm of each other, the strong nuclear forces become repulsive? Thus nuclei do not collapse.* »
- mais il n'est pas nécessaire d'être réaliste, juste esthétique.
- Principes
 - ♦ les sommets connectés par une arête devraient être dessinés proches les uns des autres
 - ♦ les sommets ne devraient pas être dessinés trop proches les uns des autres

```

for i in 1..nbIterations
  # calculate repulsive forces
  foreach v in V
    v.disp = 0
    for u in V \ {v}
      delta = v.pos - u.pos
      v.disp = v.disp + Fr(delta) * sign(delta)
  # calculate attractive forces
  foreach e in E
    delta = e.v.pos - e.u.pos
    e.v.disp = e.v.disp - Fa(delta) * sign(delta)
    e.u.disp = e.u.disp + Fa(delta) * sign(delta)
  # limits (temperature and frame) & displacements
  foreach v in V
    v.pos += sign(v.disp) * min(dispMax, v.disp)
    v.pos.x = min(xMax, max(xMin, v.pos.x))
    v.pos.y = min(yMax, max(yMin, v.pos.y))
  # decrease temperature (maximum displacement)
  dispMax = cool(dispMax)

```

- k la distance optimale entre 2 sommets

$$k = C \sqrt{\frac{\text{area}}{|v|}}$$

- Forces d'attraction et de répulsion

$$F_a(\text{distance}) = \text{distance}^2 / k$$

$$F_r(\text{distance}) = -\text{distance}^2 / k$$

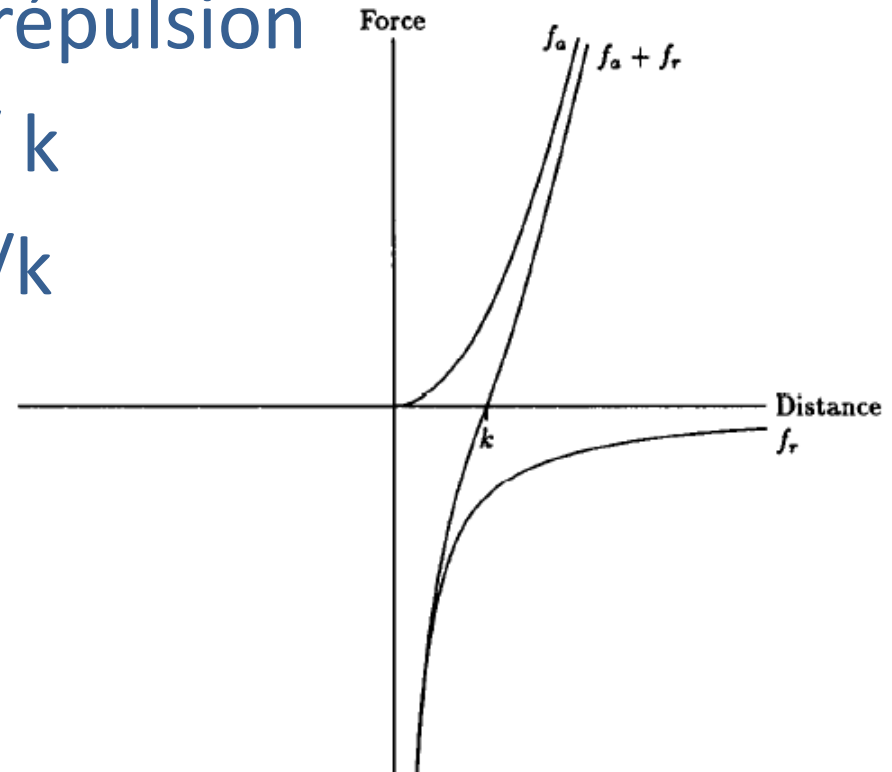


Figure 2. Forces versus distance

- A chaque itération, l'algorithme calcule les interactions entre chaque paire de sommets
- Principe :
 - ◆ placer les sommets sur une grille
 - ◆ calculer les interactions entre sommets proches (distance $< 2k$)

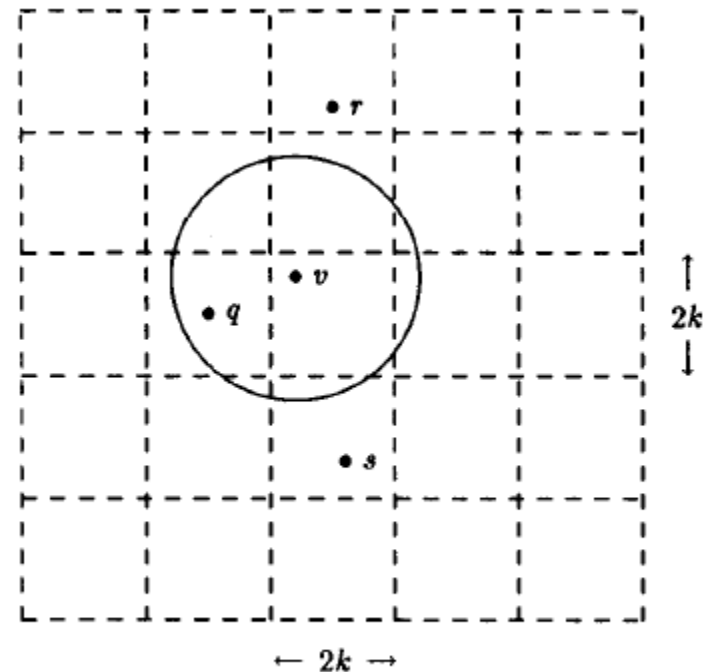


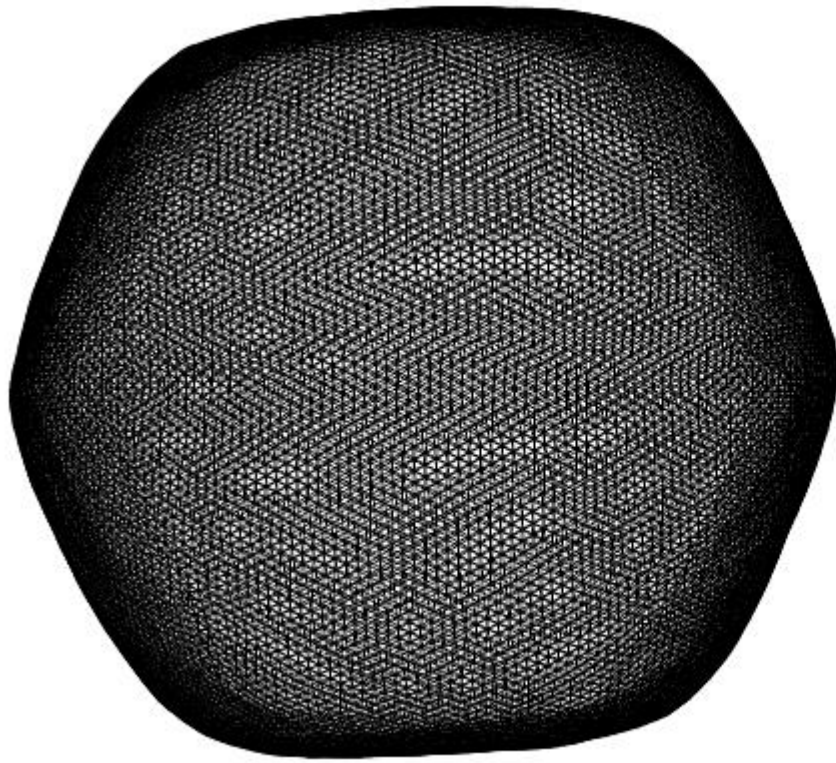
Figure 7. Calculating repulsive forces using the grid-square algorithm

- A chaque itération, chaque sommet ne peut se déplacer que d'une distance maximale
- Cette distance maximale décroît au fur et à mesure des itérations
- Idée :
 - ◆ On autorise de grands déplacements en début de procédure lorsqu'on a placé les sommets au hasard
 - ◆ A mesure des itérations, le placement des sommets devrait converger, et donc on autorise des déplacements moindres
- Remarques :
 - ◆ Il est possible de partir d'une configuration non aléatoire afin de minimiser le nombre d'itérations
 - ◆ Possibilité d'interagir avec le rendu

- Principal atout : sa rapidité
- Principe :
 - ◆ Dessiner le graphe « facilement » dans un espace ayant un grand nombre de dimensions (typiquement 50)
 - ◆ Projeter ce dessin dans un espace à 2 ou 3 dimensions
 - ◆ Astuce : faire une ACP pour déterminer de quel point de vue se placer pour faire la projection

- Distance entre chaque paire de sommets
 - ◆ avec BFS ou Dijkstra par exemple
- Choix de m pivots parmi les sommets
 - ◆ ces pivots devraient être distribués uniformément
 - ◆ à chaque pivot est associé un axe
 - ◆ les coordonnées de chaque sommet sur un axe correspondent à la plus courte distance entre le sommet et le pivot
- Le premier pivot est pris au hasard
- Les suivants sont choisis de manière à maximiser leur distance à l'ensemble des pivots déjà sélectionnés

- Après l'étape précédente, on dispose des coordonnées des sommets dans un espace à m dimensions
 - ◆ en ligne : les n individus (sommets)
 - ◆ en colonne : les coordonnées (variables) pour les m axes (pivots)
- A partir d'une analyse en composantes principales, les vecteurs propres et leur valeur propre associée fournissent la transformation à appliquer pour obtenir les coordonnées des sommets dans un espace à 2 ou 3 dimensions.



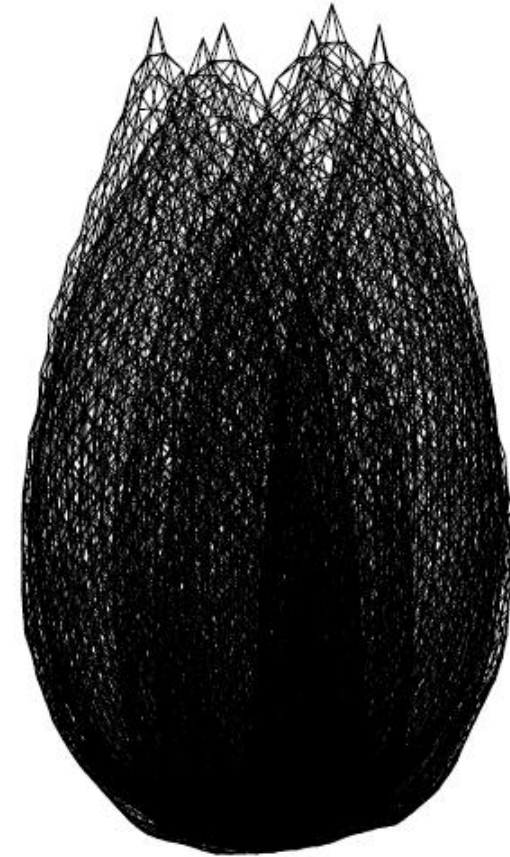
(a)

2 premières composantes principales (CP)



(b)

3^{ème} et 4^{ème} CP



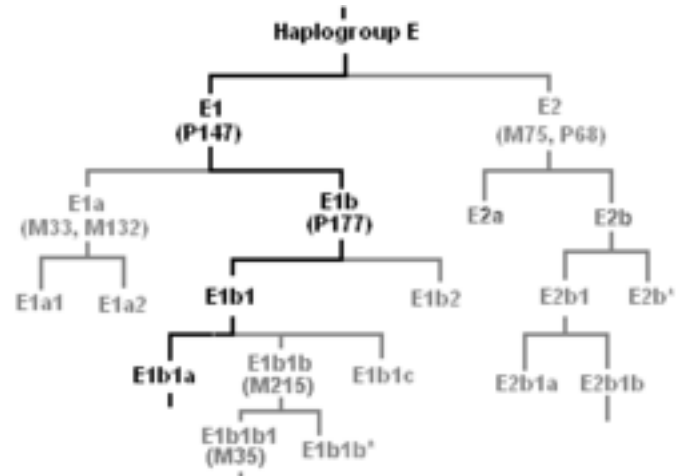
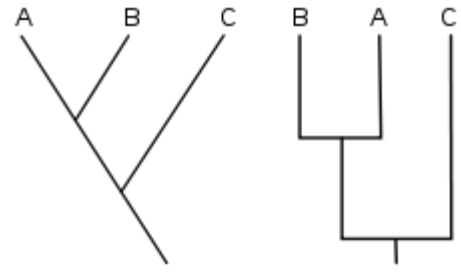
(c)

4^{ème} et 5^{ème} CP

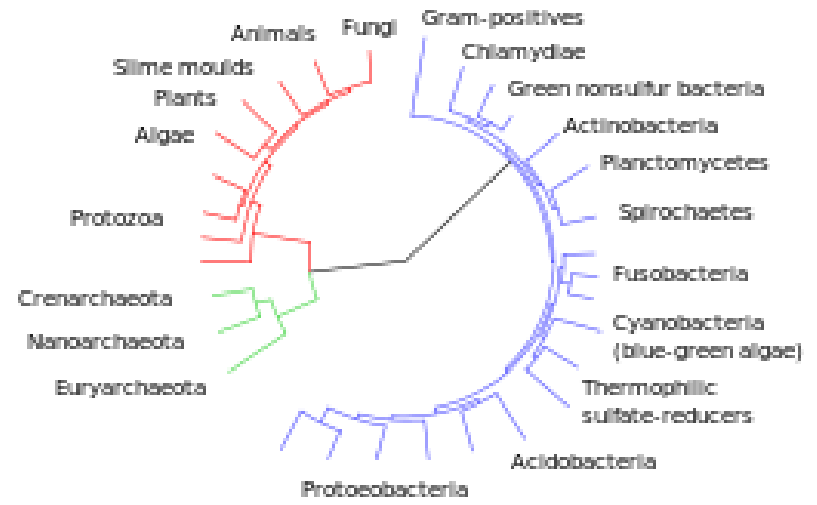
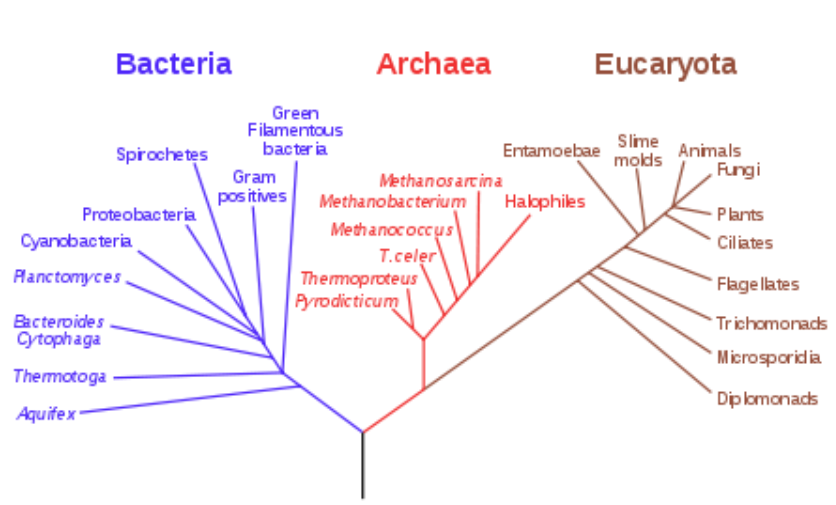
- Expression parenthésée : format **Newick** ou **New Hampshire** d'où l'extension de fichier typiquement **.nw** ou **.nhx**
- Exemples :

Etiquettes	Distances	Exemple
		(, , (,)) ;
Feuilles seulement		(A , B , (C , D)) ;
Tous		(A , B , (C , D) E) F ;
	Tous sauf la racine	(: 0 . 1 , : 0 . 2 , (: 0 . 3 , : 0 . 4) : 0 . 5) ;
	Tous	(: 0 . 1 , : 0 . 2 , (: 0 . 3 , : 0 . 4) : 0 . 5) : 0 . 0 ;
Feuilles seulement	Tous sauf la racine	(A : 0 . 1 , B : 0 . 2 , (C : 0 . 3 , D : 0 . 4) : 0 . 5) ;
Tous	Tous sauf la racine	(A : 0 . 1 , B : 0 . 2 , (C : 0 . 3 , D : 0 . 4) E : 0 . 5) F ;
Tous	Tous sauf la racine	((B : 0 . 2 , (C : 0 . 3 , D : 0 . 4) E : 0 . 5) F : 0 . 1) A ;

- Cladogrammes (pas de longueur de branche)



- Phylogrammes



source : Wikipedia

- Généralement : dessin récursif à partir de la racine (ou d'une branche interne servant de racine).
- Répartition des feuilles sur l'espace disponible
 - ◆ largeur ou hauteur disponible pour du non circulaire : $L/\#feuilles$
 - ◆ angle pour du circulaire : $A/\#feuilles$
- Tracé de la branche (selon sa longueur ou selon la profondeur dans l'arbre)
- Tracé de l'arc ou du rateau
- Appel récursif sur les descendants