# Recherche de communautés dans les graphes

## Contexte biologique

La grande disponibilité de données issues de séquençage de génomes complets rend les approches basées sur la génomique comparative particulièrement attractives pour aborder les questions relatives à l'adaptation des bactéries à leur environnement. Dans ce cadre, les systèmes codés par de grandes familles de gènes paralogues, comme les transporteurs ABC, sont ceux qui présentent le plus grand potentiel. En effet, ces systèmes sont impliqués dans l'import ou l'export de molécules diverses à travers la membrane cytoplasmique. Ils sont ubiquitaires chez les archaea et les bactéries, ils seraient donc apparus très tôt dans l'évolution. Cependant, ils peuvent avoir des distributions différentes même entre génomes apparentés suggérant des vagues successives de gains/pertes de gènes.

Les **transporteurs ABC** peuvent être classés en une trentaine de sous-familles aux quelles nous pouvons associer des types de substrats de nature spécifiques (sucres, acides aminées, peptides, ions, ferrichromes...). Cependant, nous disposons de très peu d'informations sur la nature exacte de la molécule transportée par un transporteur. Or cette information est nécessaire si l'on veut pouvoir étudier la contribution de ces transporteurs à l'adaptation des bactéries à leur environnement. Une approche bioinformatique pour tenter de répondre à cette question consiste à regrouper les différents systèmes en groupe de gènes orthologues. Sous l'hypothèse que ces gènes orthologues ont conservé la même fonction, et si le substrat transporté par le produit d'un des gènes d'un groupe a été identifié expérimentalement, alors on peut propager cette annotation à tous les autres membres du groupe. Une autre approche consiste a exploiter le contexte génomique. En effet, une forte association entre les gènes appartenant au même groupe d'orthologues et un autre gène peut révéler un lien fonctionnel entre le transporteur et le produit de ce gène, comme par exemple, si le gène code pour une enzyme, alors on peut penser que le substrat de cette enzyme a de forte chance d'être la molécule prise en charge par le transporteur ABC.

Cette approche nécessite comme préalable l'identification de groupes de **gènes orthologues**. Si cela peut paraitre assez facile à réaliser dans le cas de gènes présentant pas ou peu de duplication dans les génomes, la tache est beaucoup plus difficile à effectuer avec des familles de gènes fortement paralogues. Néanmoins, dans tous les cas, il est nécessaire d'établir de façon précise les liens de parenté entre les séquences. Cela peu se faire à l'aide de représentation arborée ou à l'aide de graphes. Les arbres fournissent une bonne modélisation des trajectoires évolutives des séquences mais sont très sensibles à la qualité des données utilisées pour les construire. A l'inverse, les **graphes** sont plus difficiles à manipuler et à interpréter, mais présentent l'avantage d'être beaucoup plus robustes et de pouvoir inclure un très grand nombre de données. Généralement, les gènes (ou les protéines) constituent les sommets du graphe. Ils sont reliés par des arcs pouvant traduire une relation évolutive entre les gènes. Ces liens sont qualitatifs (homologie, orthologie, paralogie). Si les données sont suffisamment dissemblables, le graphe peut se décomposer en un ensemble de **composantes connexes** (ensemble de sommets connectés par au moins un arc), on a alors une classification de type COG. Sinon, les groupes de gènes orthologues ne sont pas déconnectés dans le graphe, mais correspondent à des régions de plus forte densité (**communautés**). L'identification de groupes de gènes orthologues revient alors à l'identification de communautés dans un graphe.

# PRISE EN MAIN DU GRAPHE

>graph = simplify(graph)

*Quelle est l'effet de cette fonction ?
*Discutez cette observation en la ramenant à la façon dont a été construit le graphe.

Cette fonction permet de supprimer les arêtes multiples (liens d'isorthologies) entre deux sommets. Ainsi, la moitié des arêtes du graphe sont supprimées (Le graphe d'origine codait les relations A->B et B->A).

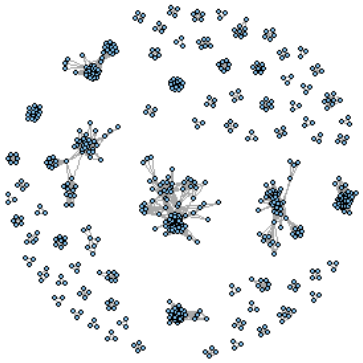
Nous utiliserons par la suite le graphe simplifié.

# TAUX DE PARALOGIE

*Quelle peut être l'origine de ce taux élevé da paralogie ?

Après avoir codé la fonction donnant le taux de paralogie et l'avoir appliquée au graphe, les résultats montrent des taux de paralogies élevés. Ceci s'explique simplement par le fait que le jeu de données présente de nombreux gènes pour peu de souches.

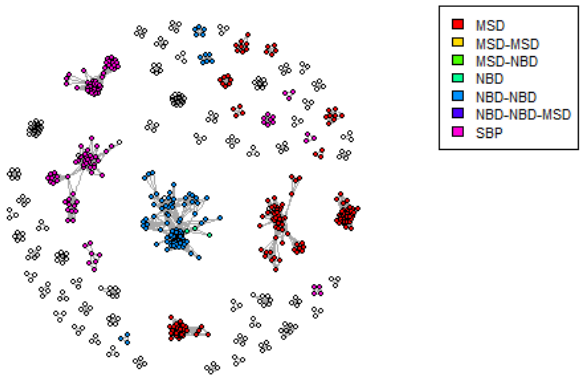
# REPRESENTATION GRAPHIQUE



Le plot montre un nombre conséquent de sous graphes isolés. Donc c'est un graphe peu dense, peu connecté avec beaucoup de composantes connexes (de tailles variables)

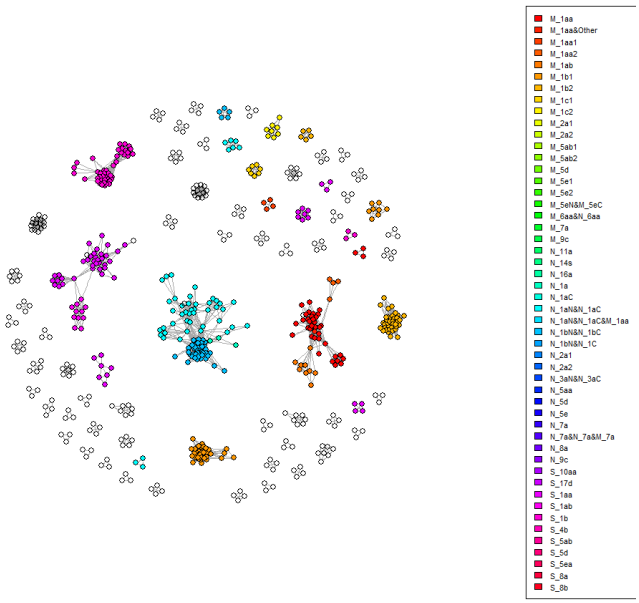
On peut penser que ces petites composantes connexes représentent les protéines se situant en amont et en aval de nos protéines du transporteur ABC.

# ANNOTATION EN DOMAINE, SOUS FAMILLES




*Annotation du graphe en domaine*

Il peut être utile de remarqué qu'à l'intérieur même d'un domaine nous observons une diversité assez importante au niveau des sous-graphes.




*Annotation du graphe en sous familles*

Cette annotation du graphe en sous famille est assez peu pertinente car il y a beaucoup de sous familles et la colorisation n'est pas celle attendue.

# COMPOSANTES CONNEXES

## * Combien de composantes connexes avez-vous obtenu ?

La fonction decompose.graph() permet d'obtenir 26 composantes connexes sur le graphe étudié.

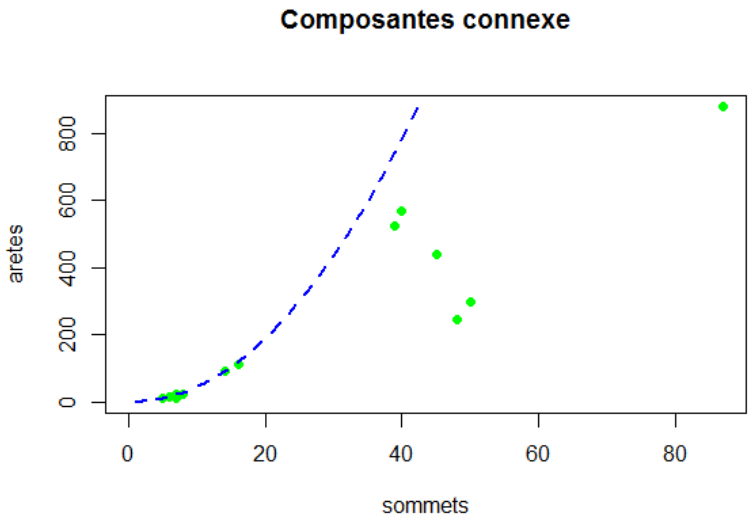
# PROPRIETES DES COMPOSANTES CONNEXES

## * Quelle est le nombre d'arêtes maximum possible dans un graphe de *n* sommets ?

La formule permettant de calculer le nombre maximum d'arêtes dans un graphe de n sommets est :

$$(n^2-n)/2$$

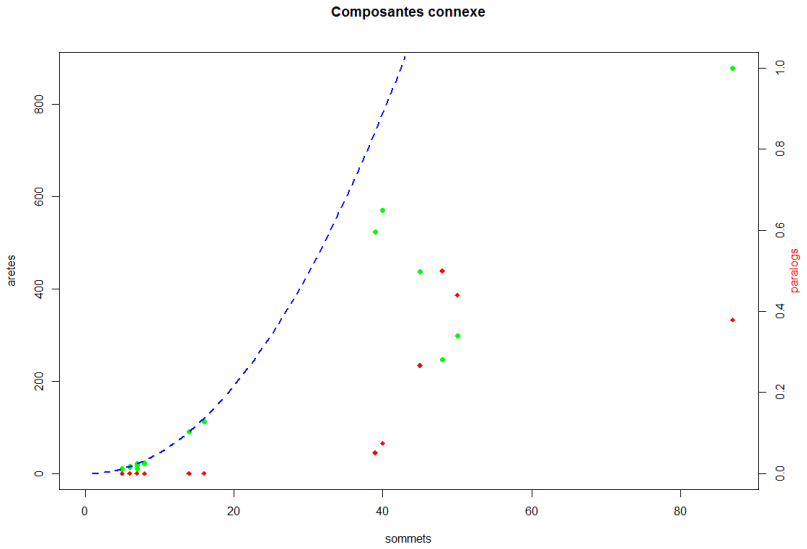
Dans la suite, il est demandé de comparer la distribution des arêtes des différentes composantes connexes en fonction de leur nombre de sommets avec la distribution maximale d'arêtes dans le cas de graphe complet. (Tous les sommets sont connectés entre eux → nombre maximal d'arêtes)




*Représentation du nombre d'arêtes des conmposantes connexes en F(#sommets)*

On ne peut pas comparer toutes les composantes connexes entre elles, seulement quelques unes suivent le maximum d'arêtes par sommet (voir figure ci-dessus). Les composantes connexes situées sur la courbes bleue sont donc des cliques et il est inutile de chercher des communautés dans celles ci. En revanche, les graphes qui s'éloignent de la distribution max sont intéressants car on va pouvoir trouver des communautés du fait qu'ils sont peu denses.

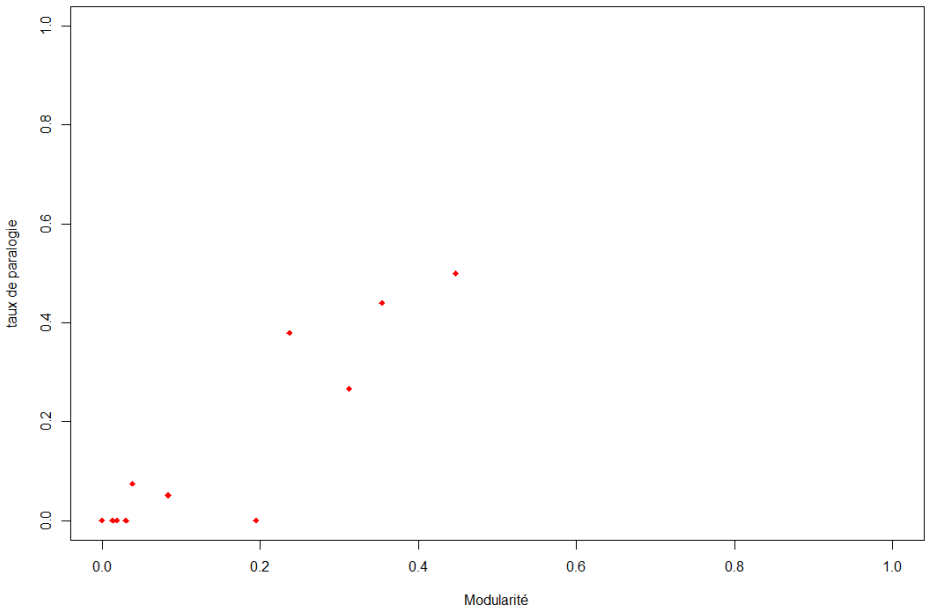
Pour compléter l'analyse, nous allons ajouter le taux de paralogies de chaque CC.




*Taux de paralgoies en fonction du nombre d'arêtes et de sommets sur les différentes composantes connexes*

Lorsque l'on se rapproche de la distribution de densité maximale, on remarque que le taux de paralogie a tendance à être assez bas voire nul. Plus on s'éloigne, plus il augmente Ceci est vrai jusqu'à un certain nombre de sommets au delà duquel l'éloignement par rapport à la courbe n'est plus proportionnel au taux de paralogie (voir la valeur extrême à 90 sommets).

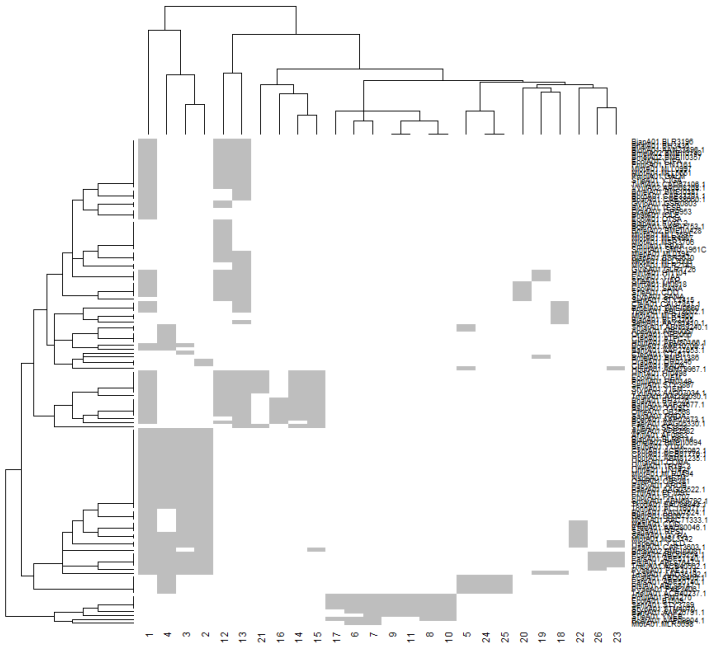
# TAUX DE PARALOGIE ET MODULARITE




*Evolution du taux de paralogie en fonction de la modularité*

La modularité est corrélée positivement avec le taux de paralogie. On observe cependant qu'il existe un ensemble de sommets ayant un taux de paralogie nul et pourtant ayant une modularité positive. Donc sans doute une structure dans ce sous graphe que l'on ne pouvait observer avec le graphique précédent. On doit ainsi pouvoir les séparer en communauté d'orthologues.

# RELATION ENTRE VOISINAGE ET DECOUPAGE EN CC




*Heatmap de la répartition des gènes dans les composantes connexes*

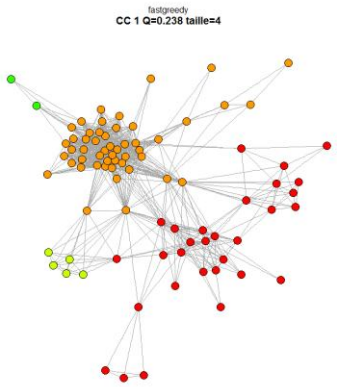
Les composantes connexes 1,3,4 sont souvent ensemble mais la composante 1 est partagée par plusieurs systèmes Il va sans doute falloir la découper car elle doit avoir beaucoup de paralogie. L'hypothèse sous jacente étant bien entendu que les gènes d'un système Co-évoluent. On voit que pour certain d'entre eux c'est le cas (1 et 3 sont souvent ensemble de même que 12 et 13). Dans la suite de ce TP, nous allons essayer de mieux découper ces composantes connexes.

# DECOMPOSITION EN COMMUNAUTES

Utilisation de l'algorithme de découpage **fastgreedy.**

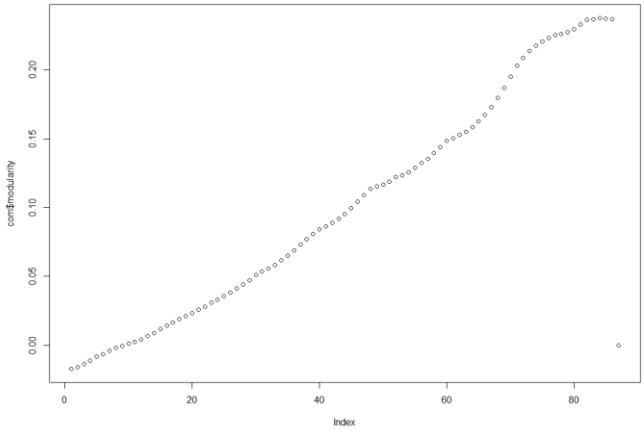
Cet algorithme utilise la modularité et améliore de manière itérative la qualité du partitionnement.

# FASTGREEDY



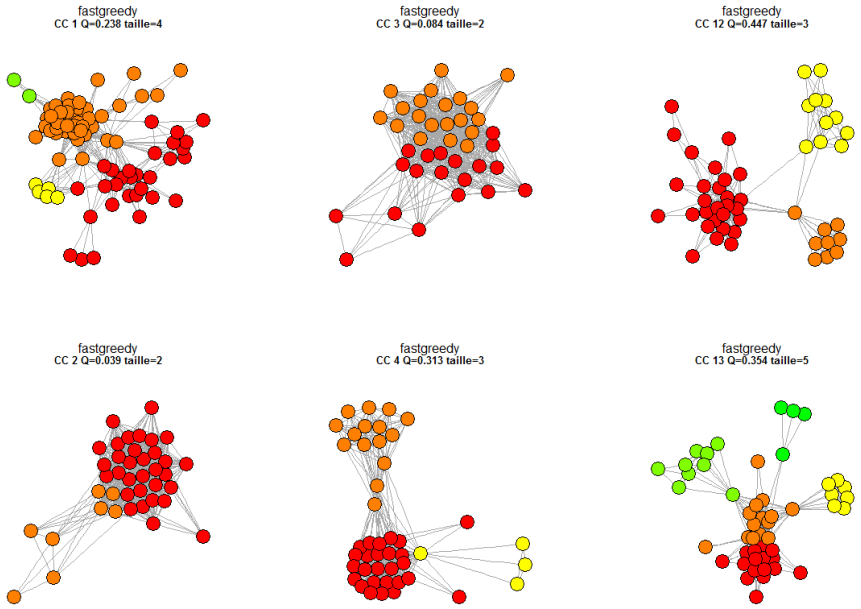

*Découpage ne communautés pour la CC_1*

On remarque que le groupe de deux vert est peut être sur-découpé. Le groupe de jaune possède des relations avec les rouges et les oranges. Finalement, c'est assez difficile à interpréter.



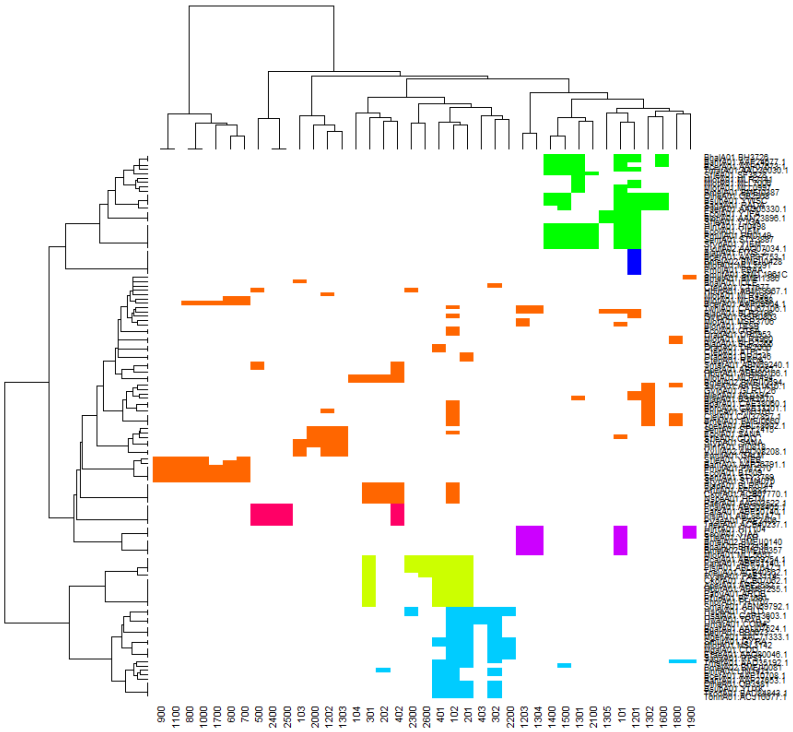

*Evolution de la modularité pas à pas*

Voyons maintenant le découpage en communauté de toutes les composantes connexes ayant un taux de paralogie supérieur à 0 :




*Découpage en communautés des CC (taux de paralogies > 0)avec la méthode fastgreedy*

Et la représentation sous forme de heatmap avec la méthode Ward et l'utilisation du coefficient de Jaccard pour le calcul de la distance :

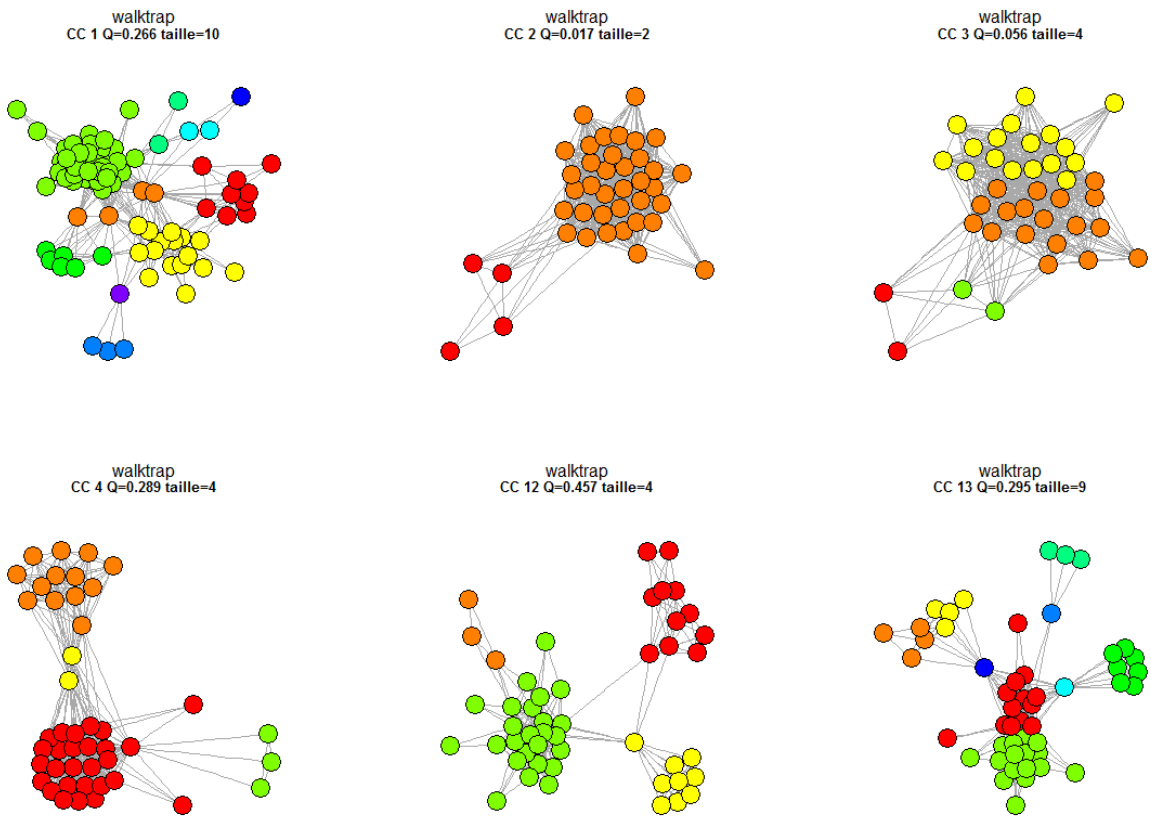


*Heatmap des CC avec la méthode de partitionnement de Ward et la distance de Jaccard*

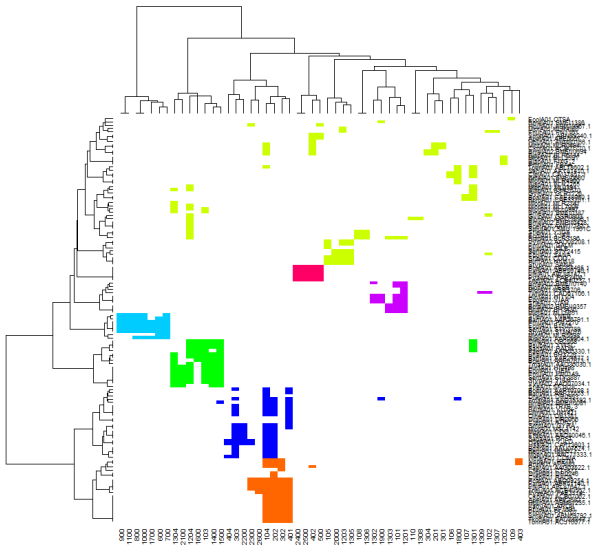
C'est encore un peu trop découpé, on retrouve des éclatements. Toutefois, on retrouve les corrélations entre partenaires. On peut arranger le sur/sous découpage de la méthode fastgreedy en modifiant les paramètres et les méthodes utilisés.

Il existe plusieurs méthodes de partitions en communautés telles que walktrap, la betweeness ou encore spinglass. Ci-dessous, les résultats de ces différentes méthodes :

# WALKTRAP

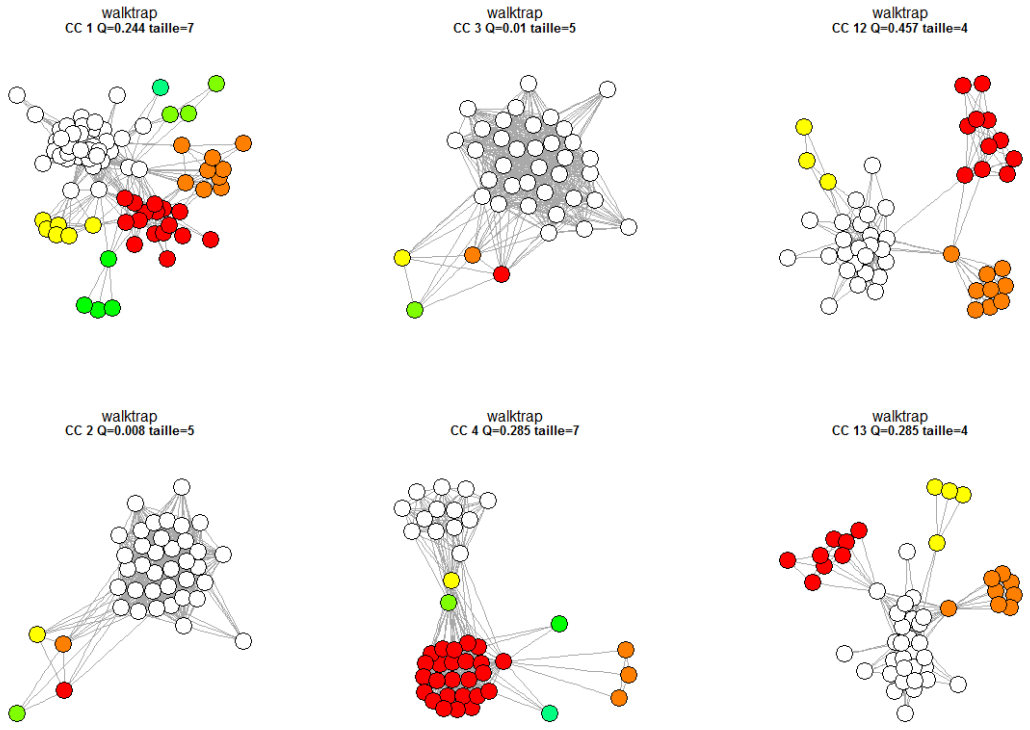



*Découpage en communautés des CC (taux de paralogies > 0)avec la méthode walktrap*



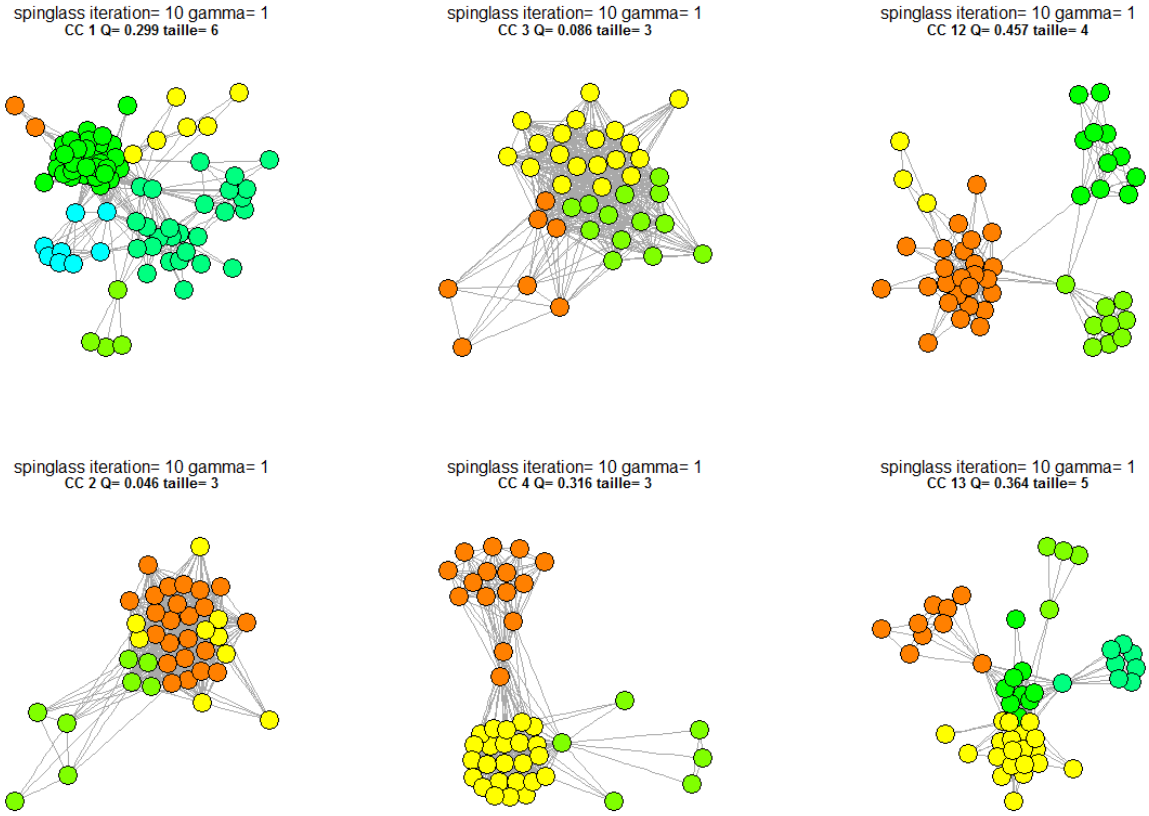
*Heatmap des CC avec la méthode de partitionnement de Ward et la distance du Phi de Pearson*

# BETWEENESS



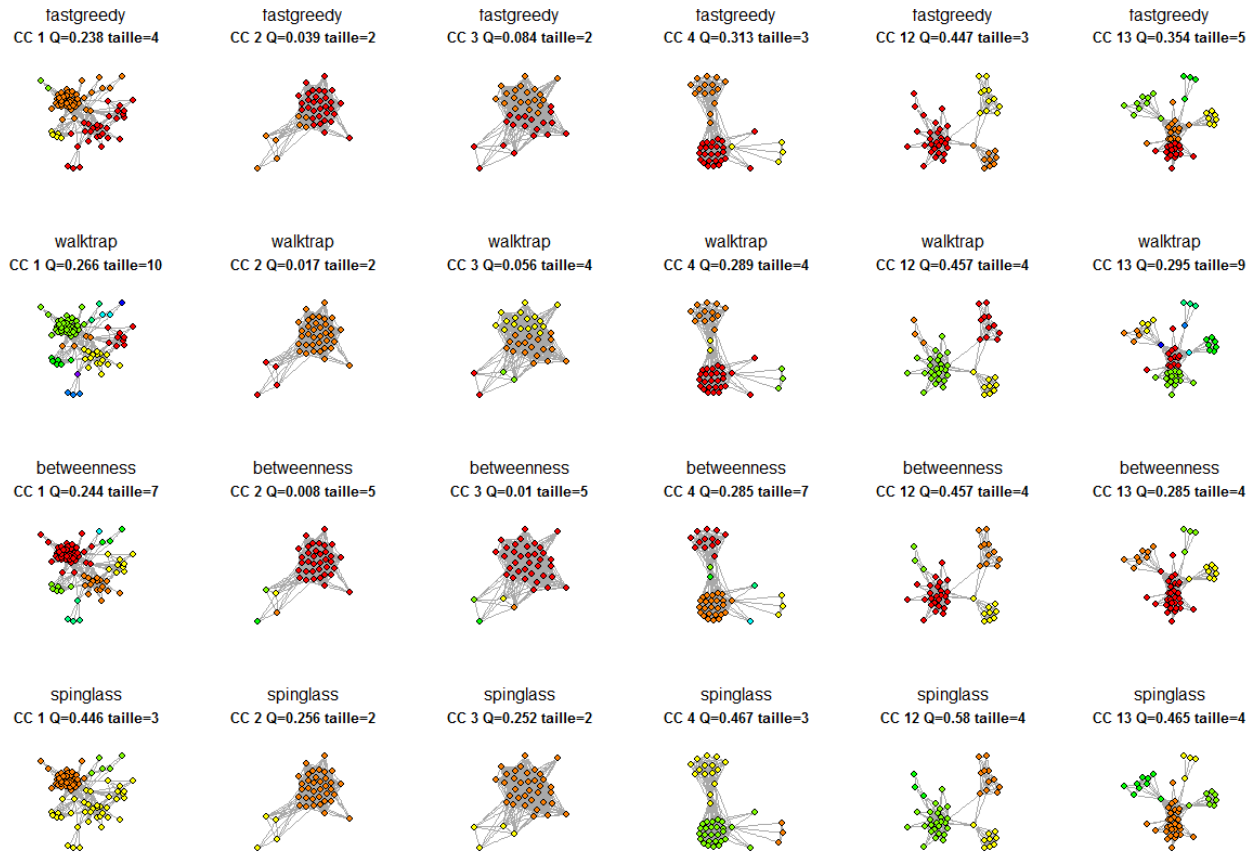

*Découpage en communautés des CC (taux de paralogies > 0)avec la méthode betweeness*

# SPINGLASS




*Découpage en communautés des CC (taux de paralogies > 0)avec la méthode spinglass*

# COMPARAISON DES METHODES




*Comparaison des découpages en communautés des CC (taux de paralogies > 0)pour toutes les méthodes*

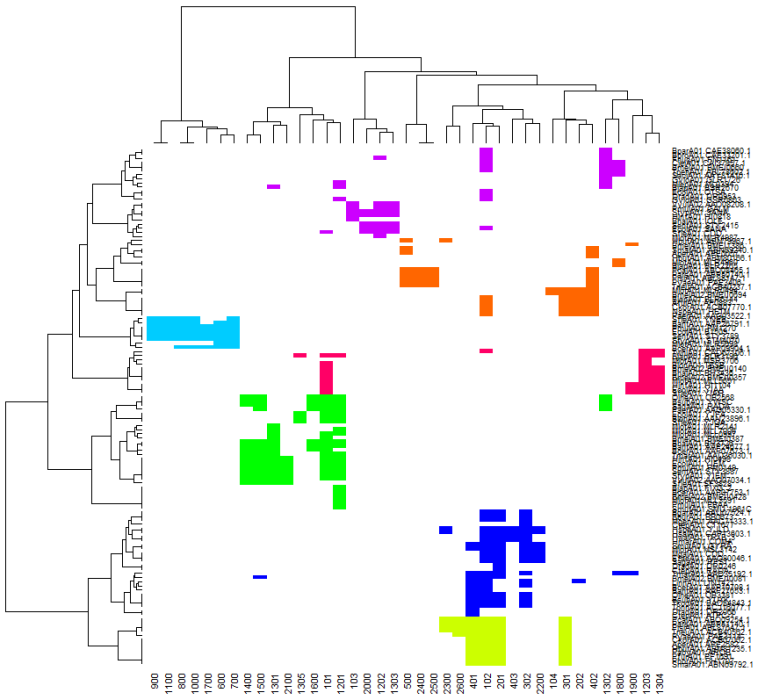
Les partitions obtenues avec les différents algorithmes ne peuvent être interprétées que de manière qualitative. Les grosses communautés sont réalisées sensiblement de la même manière quelque soit la méthode.

L'algorithme fastgreedy parait un peu limité dans la décomposition en communautés par rapport aux autres méthodes et à ce que l'on attend de ce découpage car il sur-découpe certaine communauté.

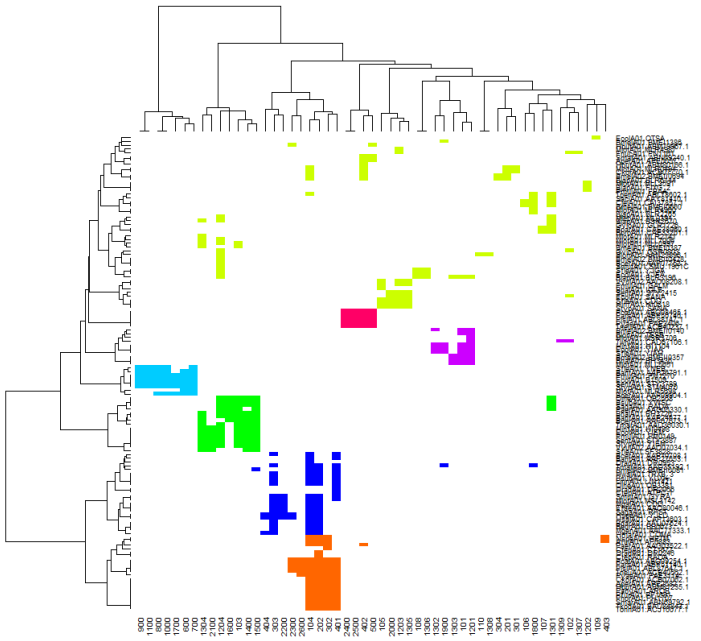
De manière générale, toutes les méthodes ont du mal à classer les sommets se trouvant aux intersections de communautés ainsi que les sommets peu connectés sauf peut être pour spinglass avec un gamma de 0.75 qui rend des résultats intéressants (au dépend d'un temps de calcul non négligeable).

Ces algorithmes sont des outils d'aides à la décision et à la recherche de gènes orthologues, mais à ce niveau de traitement, il est impossible de conclure sur leur efficacité absolue.

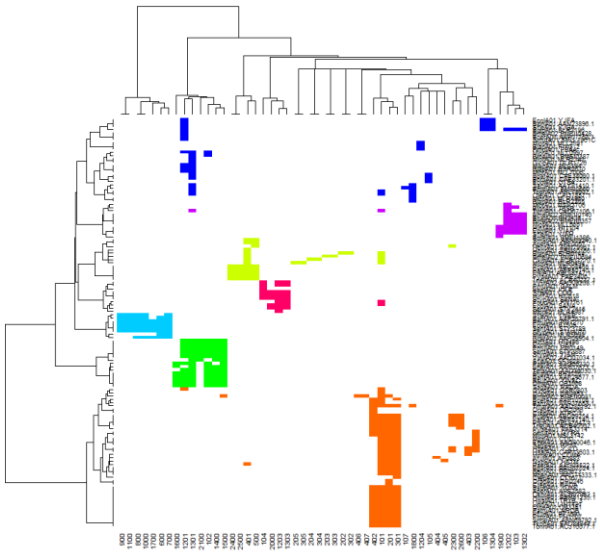
# HEATMAP



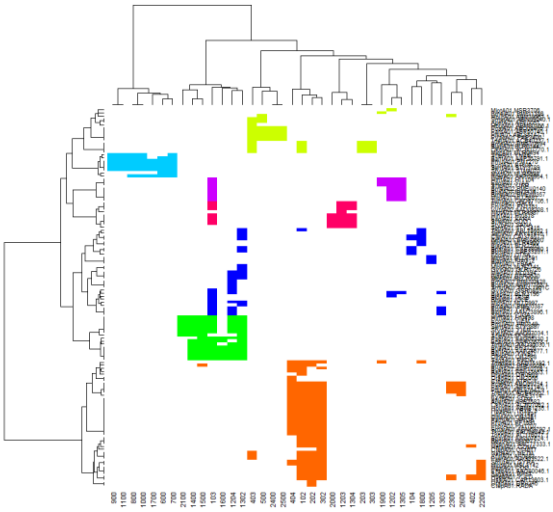
*Heatmap des CC avec la méthode de partitionnement de Ward et la distance de Jaccard(fastgreedy)*



*Heatmap des CC avec la méthode de partitionnement de Ward et la distance de Jaccard (walktrap)*



*Heatmap des CC avec la méthode de partitionnement de Ward et la distance de Jaccard (betweeness)*



*Heatmap des CC avec la méthode de partitionnement de Ward et la distance de Jaccard (spinglass)*

Visuellement, on peut voir que les méthodes spinglass et betweeness séparent mieux nos groupes mais qu'il reste encore des gènes dont on ne peut affirmer l'appartenance à un groupe.

## CROISEMENT DES PARTITIONS

Je n'ai pu réaliser que la fonction produisant la matrice de comptage mais je n'ai pas compris de quelle manière on pouvait représenter ces résultats sous forme de heatmap (les heatmap que j'obtiens sont très singulières et je n'arrive pas à les interpréter). Aussi, je joins en annexe la fonction.

# ANNEXE

## CROISEMENT DES PARTITIONS

```
classCC <- matrix[,6:9]
rownames(classCC)
#Matrice d'adjacence
adjMat <- matrix(data = 0, ncol = nrow(classCC), nrow = nrow(classCC))

for (i in 1:nrow(classCC)){
   pos1 <- which(classCC[-(1:i),1] == classCC[i,1]) + i
   pos2 <- which(classCC[-(1:i),2] == classCC[i,2]) + i
   pos3 <- which(classCC[-(1:i),3] == classCC[i,3]) + i
   pos4 <- which(classCC[-(1:i),4] == classCC[i,4]) + i
   adjMat[i,pos1] <- adjMat[i,pos1] + 1
   adjMat[i,pos2] <- adjMat[i,pos2] + 1
   adjMat[i,pos3] <- adjMat[i,pos3] + 1
   adjMat[i,pos4] <- adjMat[i,pos4] + 1
}

test <- adjMat + t(adjMat) #Pour avoir la matrice carré
head(test)

#Je ne vois pas comment utiliser une heatmap ici.
heatmap(adjMat, scale='none', col=color)
```