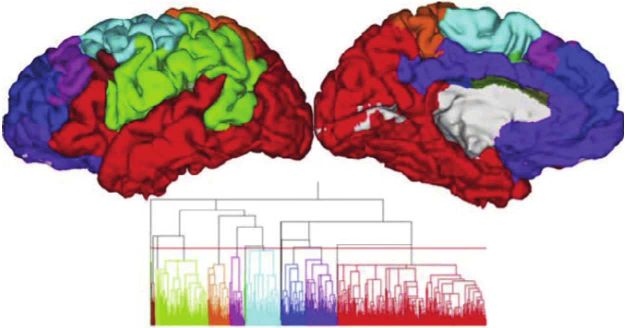
# CS 4884: Modules

T. M. Murali

February 22 and 24, 2022



# Summary of Course Thus Far

- Clustering coefficient is a local measure of graph density.
- Small world property captures global features of graph density.

# Summary of Course Thus Far

- Clustering coefficient is a local measure of graph density.
- Small world property captures global features of graph density.

Are there intermediate notions of graph density?

- We have already considered components and shortest paths.

# Why Modules?

Why should (brain) networks be modular?
Do modules exist in brain networks?
How do we define modules and find them?

# Why Modules?

Why should (brain) networks be modular?

Do modules exist in brain networks?

How do we define modules and find them?

# Why Modules?

Why should (brain) networks be modular?
Do modules exist in brain networks?
How do we define modules and find them?

- Modularity and hierarchical organisation offer several advantages: evolvability, flexibility, adaptability, and complexity (Simon, 1962).
- Focal pathology affecting a tightly interconnected brain module will be less likely to spread and affect other areas.
- Breakdown of modularity can lead to a propensity for hyper-synchronized, seizure-like dynamics.

# Why Modules?

Why should (brain) networks be modular?
Do modules exist in brain networks?
How do we define modules and find them?

- Modularity and hierarchical organisation offer several advantages: evolvability, flexibility, adaptability, and complexity (Simon, 1962).
- Focal pathology affecting a tightly interconnected brain module will be less likely to spread and affect other areas.
- Breakdown of modularity can lead to a propensity for hyper-synchronized, seizure-like dynamics.

# Why Modules?

Why should (brain) networks be modular?
Do modules exist in brain networks?
How do we define modules and find them?

- Modularity and hierarchical organisation offer several advantages: evolvability, flexibility, adaptability, and complexity (Simon, 1962).
- Focal pathology affecting a tightly interconnected brain module will be less likely to spread and affect other areas.
- Breakdown of modularity can lead to a propensity for hyper-synchronized, seizure-like dynamics.
- Do E-R graphs contain modules?

# Why Modules?

Why should (brain) networks be modular?
Do modules exist in brain networks?
How do we define modules and find them?

- Modularity and hierarchical organisation offer several advantages: evolvability, flexibility, adaptability, and complexity (Simon, 1962).
- Focal pathology affecting a tightly interconnected brain module will be less likely to spread and affect other areas.
- Breakdown of modularity can lead to a propensity for hyper-synchronized, seizure-like dynamics.
- Do E-R graphs contain modules? No, because all nodes have roughly the same degree.

# Why Modules?

Why should (brain) networks be modular?
Do modules exist in brain networks?
How do we define modules and find them?

- Modularity and hierarchical organisation offer several advantages: evolvability, flexibility, adaptability, and complexity (Simon, 1962).
- Focal pathology affecting a tightly interconnected brain module will be less likely to spread and affect other areas.
- Breakdown of modularity can lead to a propensity for hyper-synchronized, seizure-like dynamics.
- Do E-R graphs contain modules? No, because all nodes have roughly the same degree.
- Do W-S graphs contain modules?

# Why Modules?

Why should (brain) networks be modular?
Do modules exist in brain networks?
How do we define modules and find them?

- Modularity and hierarchical organisation offer several advantages: evolvability, flexibility, adaptability, and complexity (Simon, 1962).
- Focal pathology affecting a tightly interconnected brain module will be less likely to spread and affect other areas.
- Breakdown of modularity can lead to a propensity for hyper-synchronized, seizure-like dynamics.
- Do E-R graphs contain modules? No, because all nodes have roughly the same degree.
- Do W-S graphs contain modules? No, although other small-world networks can contain modules.

# Why Modules?

Why should (brain) networks be modular?
Do modules exist in brain networks?
How do we define modules and find them?

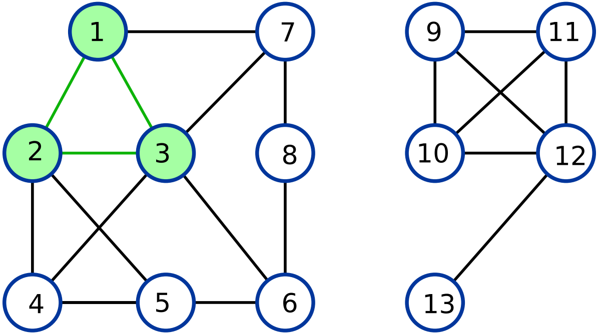
- Modularity and hierarchical organisation offer several advantages: evolvability, flexibility, adaptability, and complexity (Simon, 1962).
- Focal pathology affecting a tightly interconnected brain module will be less likely to spread and affect other areas.
- Breakdown of modularity can lead to a propensity for hyper-synchronized, seizure-like dynamics.
- Do E-R graphs contain modules? No, because all nodes have roughly the same degree.
- Do W-S graphs contain modules? No, although other small-world networks can contain modules.
- But the brain is indeed modular: organ, hemispheres, coarse divisions, lobes, cytoarchitectural areas, nuclei, etc.

# Why Modules?

Why should (brain) networks be modular?
Do modules exist in brain networks?
How do we define modules and find them?

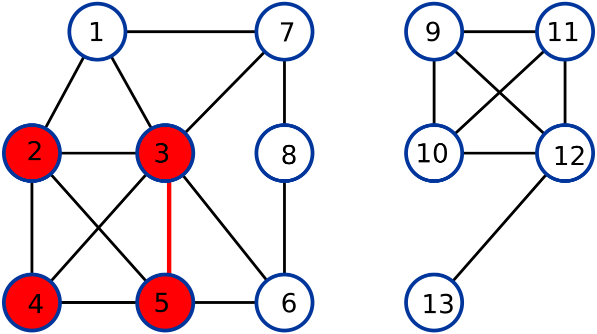
- Modularity and hierarchical organisation offer several advantages: evolvability, flexibility, adaptability, and complexity (Simon, 1962).
- Focal pathology affecting a tightly interconnected brain module will be less likely to spread and affect other areas.
- Breakdown of modularity can lead to a propensity for hyper-synchronized, seizure-like dynamics.
- Do E-R graphs contain modules? No, because all nodes have roughly the same degree.
- Do W-S graphs contain modules? No, although other small-world networks can contain modules.
- But the brain is indeed modular: organ, hemispheres, coarse divisions, lobes, cytoarchitectural areas, nuclei, etc.

# Defining Modules



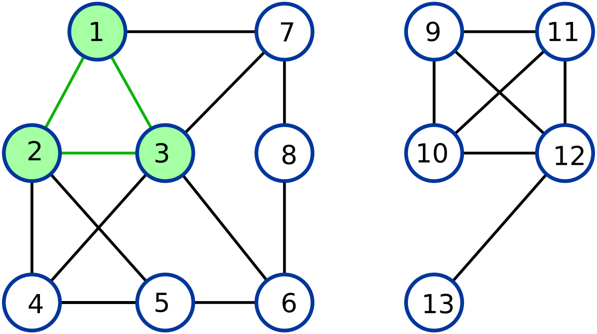

- How do we define a module in an undirected graph?
- In an undirected graph $G = (V, E)$, a subset of nodes $C \subseteq V$ is a *clique* or *complete subgraph* if for every pair of nodes $u, v \in C$, $(u, v)$ is an edge in $E$.

# Defining Modules



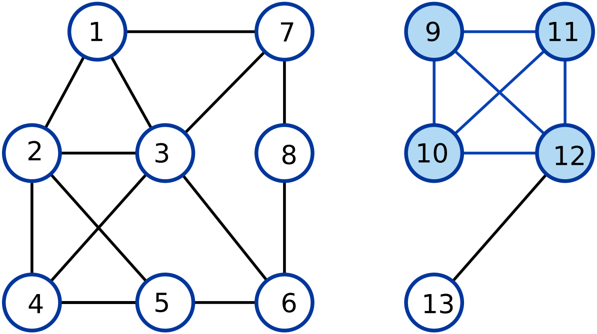

- How do we define a module in an undirected graph?
- In an undirected graph $G = (V, E)$, a subset of nodes $C \subseteq V$ is a *clique* or *complete subgraph* if for every pair of nodes $u, v \in C$, $(u, v)$ is an edge in $E$.

# Defining Modules



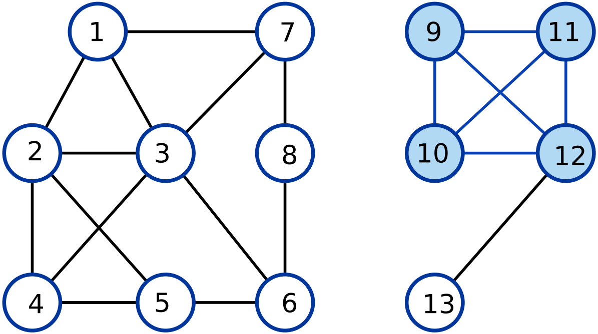

- How do we define a module in an undirected graph?
- In an undirected graph $G = (V, E)$, a subset of nodes $C \subseteq V$ is a *clique* or *complete subgraph* if for every pair of nodes $u, v \in C$, $(u, v)$ is an edge in $E$.
  - A clique $C$ is *maximal* if no node outside $C$ can be added to it, i.e., for every node $x \in V - C$, $x$ is not connected to at least one node in $C$.

# Defining Modules




- How do we define a module in an undirected graph?
- In an undirected graph $G = (V, E)$, a subset of nodes $C \subseteq V$ is a *clique* or *complete subgraph* if for every pair of nodes $u, v \in C$, $(u, v)$ is an edge in $E$.
  - A clique $C$ is *maximal* if no node outside $C$ can be added to it, i.e., for every node $x \in V - C$, $x$ is not connected to at least one node in $C$.
  - A clique $C$ is *maximum* if there is no clique $C'$ in $G$ with more nodes than $C$.

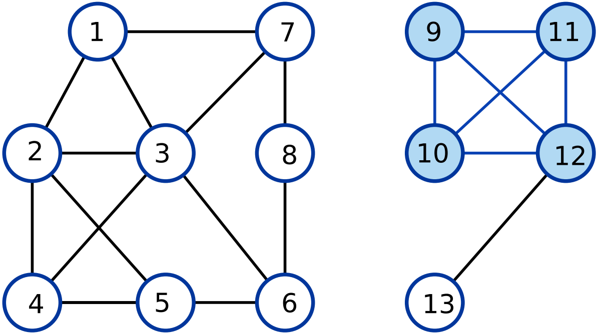
# Computing a Maximum Clique




Maximum Clique

Given an undirected, unweighted graph $G(V, E)$, compute the largest clique in $G$.

# Computing a Maximum Clique

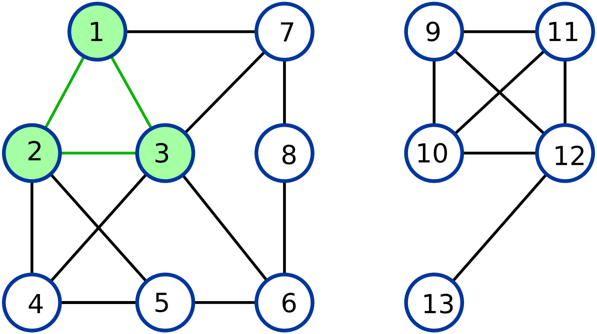



Maximum Clique

Given an undirected, unweighted graph $G(V, E)$,
compute the largest clique in $G$.

- Computing a maximum clique is NP-hard.
- Any algorithm that can provably compute the maximum clique is likely to have a running time that is exponential in the size of the graph.

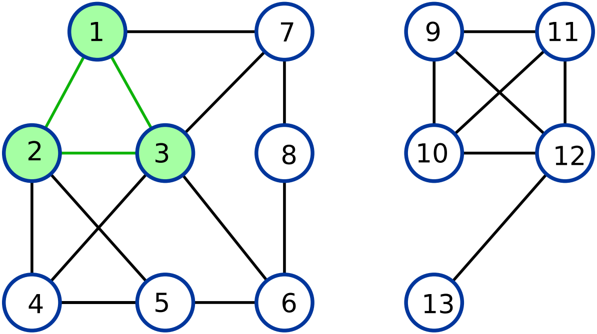
# Computing a Maximal Clique




Maximal Clique

Given an undirected, unweighted graph $G(V, E)$, compute a maximal clique in $G$.

# Computing a Maximal Clique

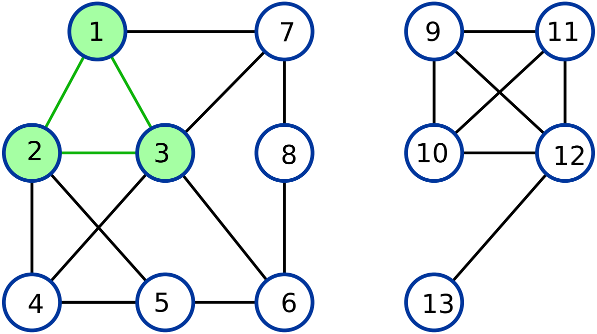



Maximal Clique

Given an undirected, unweighted graph $G(V, E)$, compute a maximal clique in $G$.

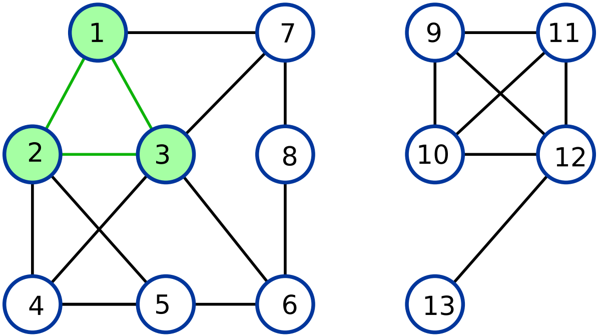
1. Select an arbitrary node $v$ and add it to $S$ (the clique we will output).
2. If there is a node $u$ in $V - S$ that is connected to every node in $S$, add $u$ to $S$.
3. Repeat the previous step until no such node $u$ is found.

# Running Time to Compute a Maximal Clique



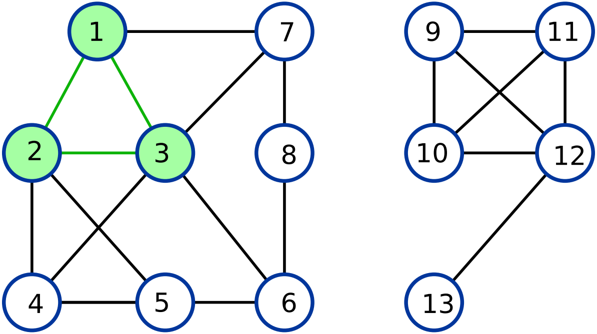

1. Select an arbitrary node $v$ and add it to $S$ (the clique we will output).
2. If there is a node $u$ in $V - S$ that is connected to every node in $S$, add $u$ to $S$.
3. Repeat the previous step until no such node $u$ is found.

# Running Time to Compute a Maximal Clique



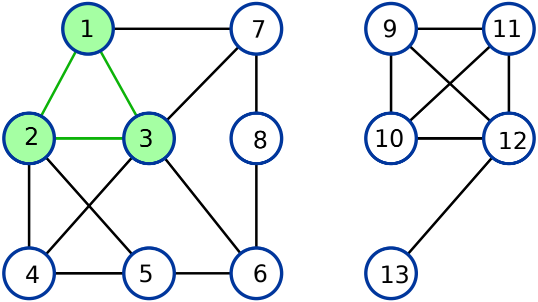

1. Select an arbitrary node $v$ and add it to $S$ (the clique we will output).
2. If there is a node $u$ in $V - S$ that is connected to every node in $S$, add $u$ to $S$. $O(n|S|)$ checks for edge existence.
3. Repeat the previous step until no such node $u$ is found.

# Running Time to Compute a Maximal Clique



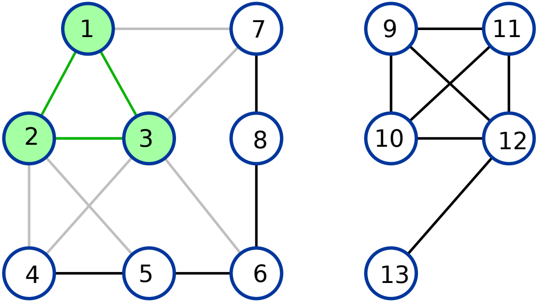

1. Select an arbitrary node $v$ and add it to $S$ (the clique we will output).
2. If there is a node $u$ in $V - S$ that is connected to every node in $S$, add $u$ to $S$. $O(n|S|)$ checks for edge existence.
3. Repeat the previous step until no such node $u$ is found. $O(n|S|^2)$ checks for edge existence.

# Clique Decomposition



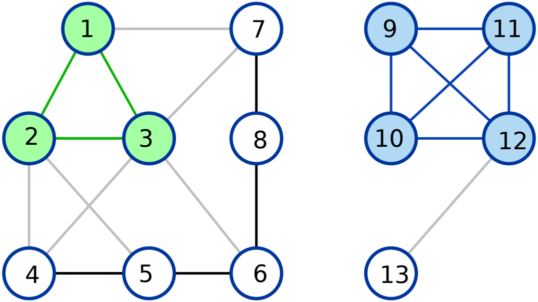

- What do we do after computing a maximal clique?

# Clique Decomposition



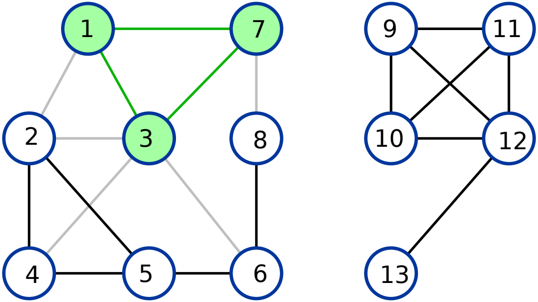

- What do we do after computing a maximal clique?
- Delete nodes in that clique from the graph and repeat.

# Clique Decomposition



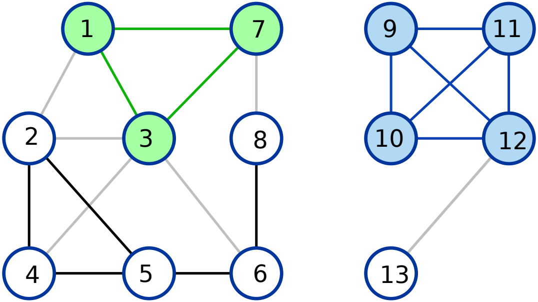

- What do we do after computing a maximal clique?
- Delete nodes in that clique from the graph and repeat.
- The resulting set of cliques forms a *clique decomposition* of $G$.

# Clique Decomposition



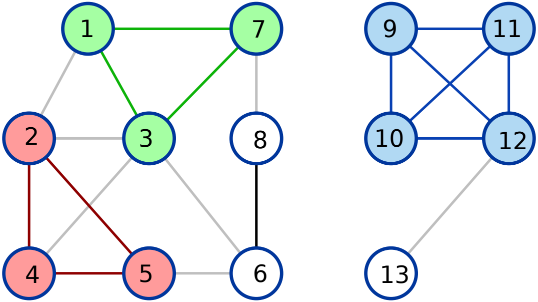

- What do we do after computing a maximal clique?
- Delete nodes in that clique from the graph and repeat.
- The resulting set of cliques forms a *clique decomposition* of $G$.
- Sequence of cliques found depends on order of processing nodes.

# Clique Decomposition



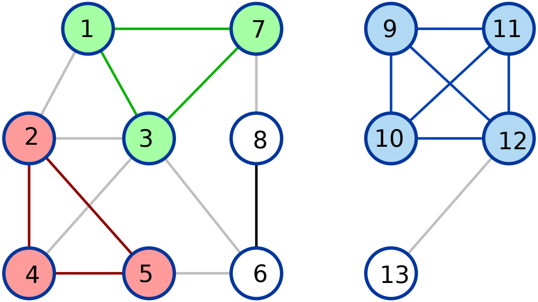

- What do we do after computing a maximal clique?
- Delete nodes in that clique from the graph and repeat.
- The resulting set of cliques forms a *clique decomposition* of $G$.
- Sequence of cliques found depends on order of processing nodes.

# Clique Decomposition



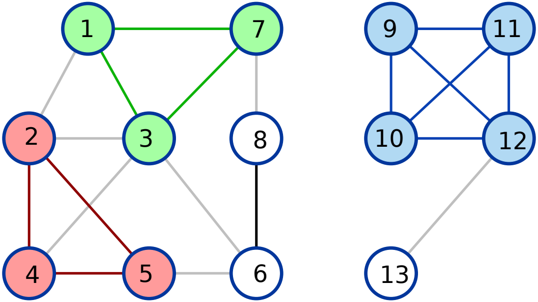

- What do we do after computing a maximal clique?
- Delete nodes in that clique from the graph and repeat.
- The resulting set of cliques forms a *clique decomposition* of $G$.
- Sequence of cliques found depends on order of processing nodes.

# Clique Decomposition



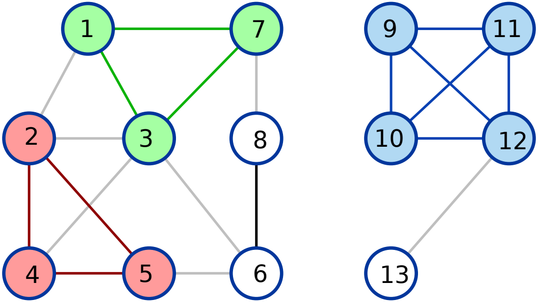

- What do we do after computing a maximal clique?
- Delete nodes in that clique from the graph and repeat.
- The resulting set of cliques forms a *clique decomposition* of $G$.
- Sequence of cliques found depends on order of processing nodes.
- There is no notion of correctness here since we defined what we compute (the clique decomposition) based on an algorithm we specified.

# Clique Decomposition



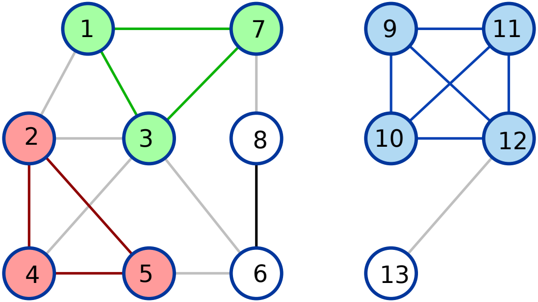

- What do we do after computing a maximal clique?
- Delete nodes in that clique from the graph and repeat.
- The resulting set of cliques forms a *clique decomposition* of $G$.
- Sequence of cliques found depends on order of processing nodes.
- There is no notion of correctness here since we defined what we compute (the clique decomposition) based on an algorithm we specified.
- Will every edge in $G$ be in some clique in the decomposition? Can a node be in multiple cliques?

# Clique Decomposition



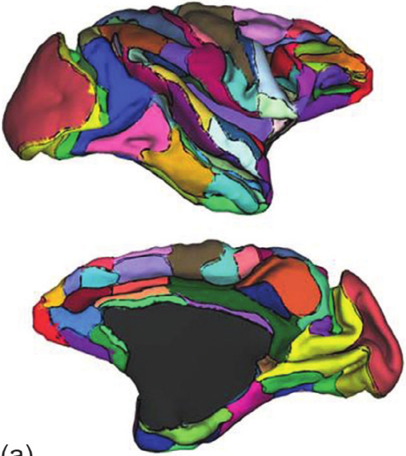

- What do we do after computing a maximal clique?
- Delete nodes in that clique from the graph and repeat.
- The resulting set of cliques forms a *clique decomposition* of $G$.
- Sequence of cliques found depends on order of processing nodes.
- There is no notion of correctness here since we defined what we compute (the clique decomposition) based on an algorithm we specified.
- Will every edge in $G$ be in some clique in the decomposition? Can a node be in multiple cliques? No, to both questions.

# Clique Decomposition




- What do we do after computing a maximal clique?
- Delete nodes in that clique from the graph and repeat.
- The resulting set of cliques forms a *clique decomposition* of $G$.
- Sequence of cliques found depends on order of processing nodes.
- There is no notion of correctness here since we defined what we compute (the clique decomposition) based on an algorithm we specified.
- Will every edge in $G$ be in some clique in the decomposition? Can a node be in multiple cliques? No, to both questions.
- Modification: After finding a clique, delete only the edges in it.

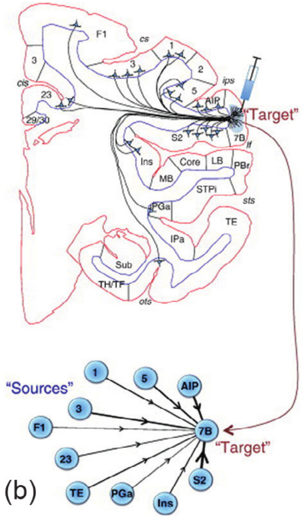
# Structural Connectivity at the Mesoscale



(a)

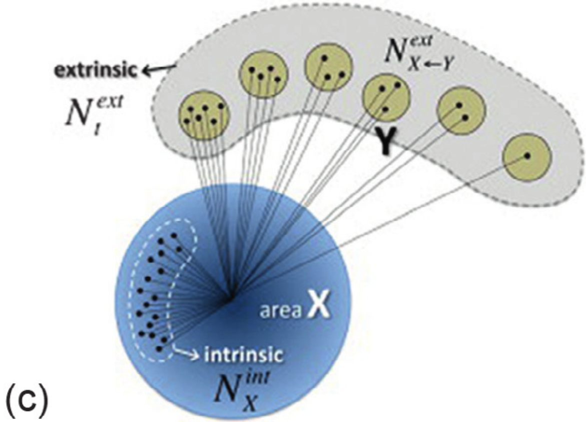
Parcellate the macaque cortex into 91 areas, defined according to cytoarchitecture and sulco-gyral landmarks.

# Structural Connectivity at the Mesoscale



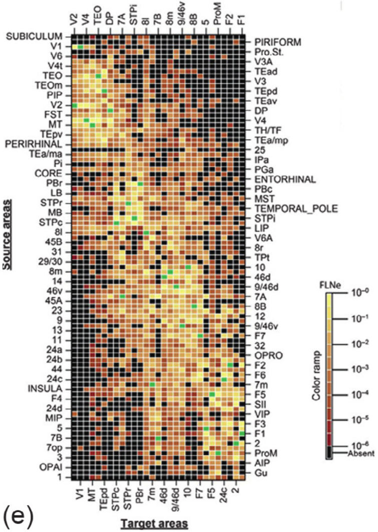

Use retrograde tract tracing. Determine edges coming into node representing area of injection from "labelled" nodes representing neurons that the tracer reaches.

# Structural Connectivity at the Mesoscale




Injection is at $X$: $w(Y, X) = \frac{\text{number of neurons labelled in } Y}{\text{total number of labelled neurons}}$

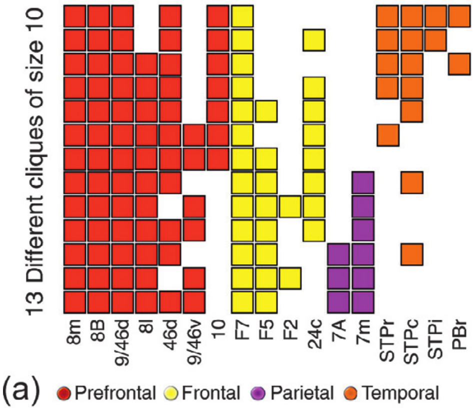
# Structural Connectivity at the Mesoscale




Example of connectivity matrix.

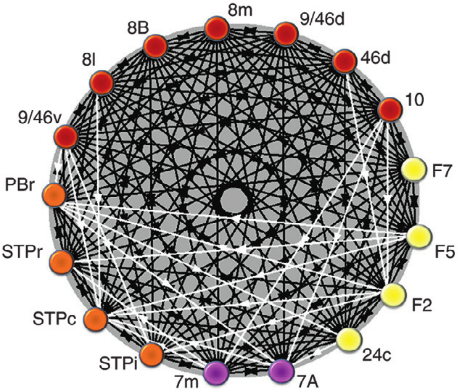
Edge weights range over six orders of magnitude.

# Cliques in Macaque Cerebral Cortex Connectome




- 29-node directed graph representing connectome of the cerebral cortex of the macaque; only considering nodes with tracer injection points.
- Computed all 13 maximum cliques, each of which had 10 nodes.

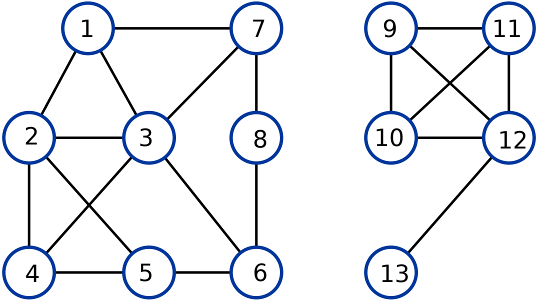
# Cliques in Macaque Cerebral Cortex Connectome




(b)

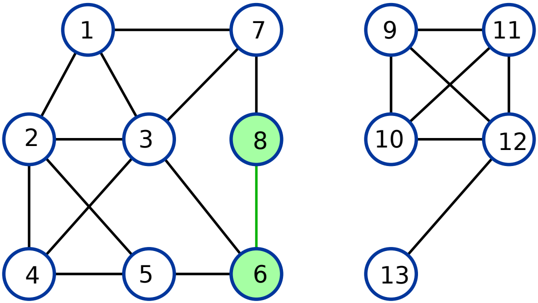
- 29-node directed graph representing connectome of the cerebral cortex of the macaque; only considering nodes with tracer injection points.
- Computed all 13 maximum cliques, each of which had 10 nodes.
- Union of cliques formed a dense subgraph among 17 nodes.

# $k$-Cores



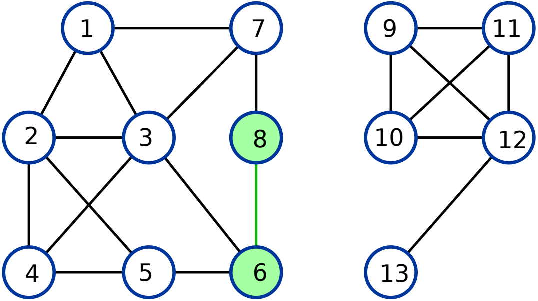

- In an undirected graph $G = (V, E)$, a subset of nodes $C \subseteq V$ is a *$k$-core* if every node $u \in C$ is connected in $G$ to at least $k$ nodes in $C$.

# $k$-Cores



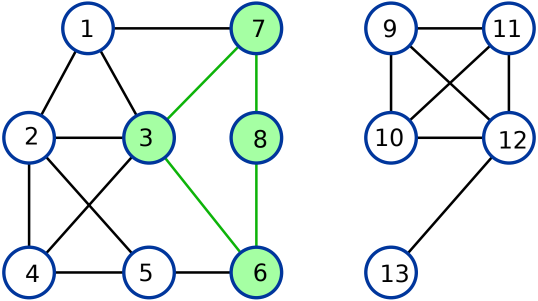

- In an undirected graph $G = (V, E)$, a subset of nodes $C \subseteq V$ is a *$k$-core* if every node $u \in C$ is connected in $G$ to at least $k$ nodes in $C$.
- What is largest the 1-core of $G$?

# $k$-Cores



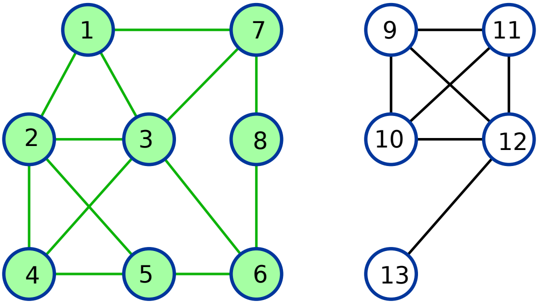

- In an undirected graph $G = (V, E)$, a subset of nodes $C \subseteq V$ is a *$k$-core* if every node $u \in C$ is connected in $G$ to at least $k$ nodes in $C$.
- What is largest the 1-core of $G$? $G$ itself (without any nodes of degree zero).

# $k$-Cores



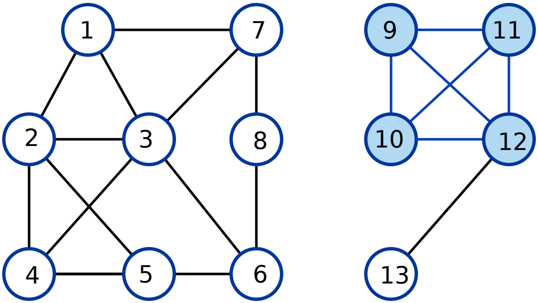

- In an undirected graph $G = (V, E)$, a subset of nodes $C \subseteq V$ is a *$k$-core* if every node $u \in C$ is connected in $G$ to at least $k$ nodes in $C$.
- What is largest the 1-core of $G$? $G$ itself (without any nodes of degree zero).

# $k$-Cores



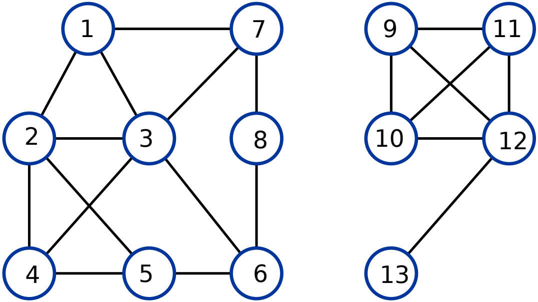

- In an undirected graph $G = (V, E)$, a subset of nodes $C \subseteq V$ is a *$k$-core* if every node $u \in C$ is connected in $G$ to at least $k$ nodes in $C$.
- What is largest the 1-core of $G$? $G$ itself (without any nodes of degree zero).

# $k$-Cores




- In an undirected graph $G = (V, E)$, a subset of nodes $C \subseteq V$ is a *$k$-core* if every node $u \in C$ is connected in $G$ to at least $k$ nodes in $C$.
- What is largest the 1-core of $G$? $G$ itself (without any nodes of degree zero).
- Does this graph have a 4-core?

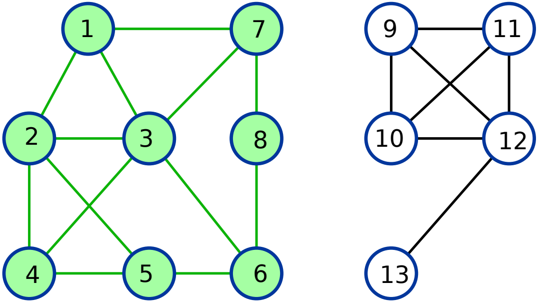
# Problems related to $k$-cores




$k$-CORE EXISTENCE

Given an undirected, unweighted graph $G(V, E)$ and an integer $k$, compute the $k$-core with the largest number of nodes in $G$, if it exists.

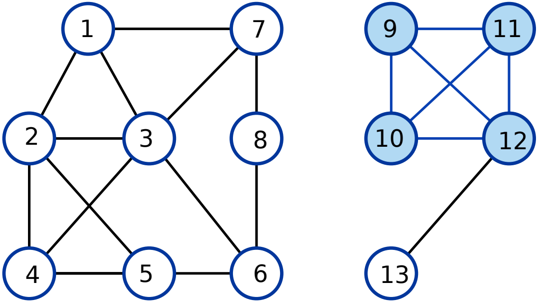
# Problems related to $k$-cores




$k$-CORE EXISTENCE

Given an undirected, unweighted graph $G(V, E)$ and an integer $k$, compute the $k$-core with the largest number of nodes in $G$, if it exists.

# Problems related to $k$-cores



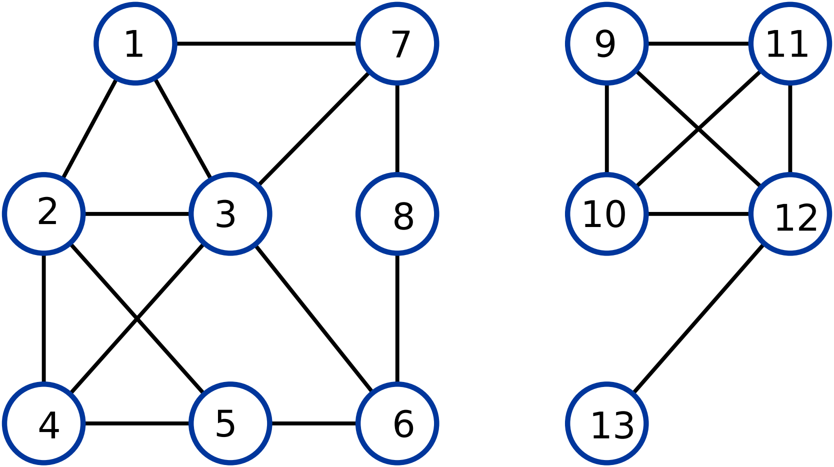

$k$-CORE EXISTENCE

Given an undirected, unweighted graph $G(V, E)$ and an integer $k$, compute the $k$-core with the largest number of nodes in $G$, if it exists.

LARGEST $k$-CORE

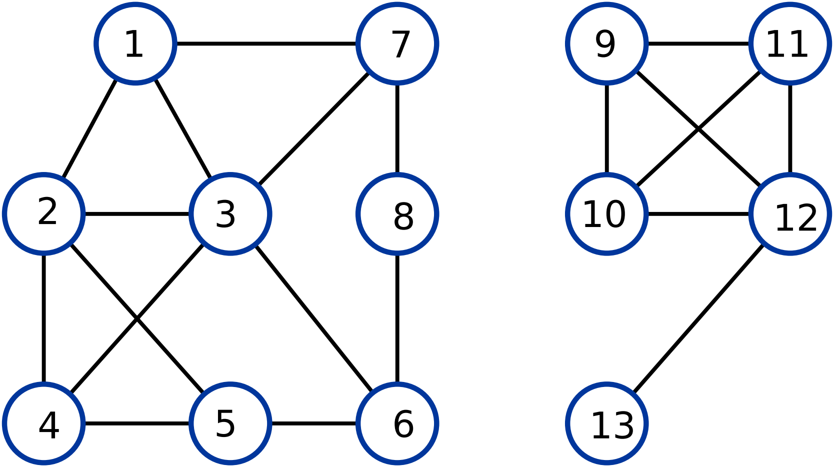
Given an undirected, unweighted graph $G(V, E)$, compute the largest value of $k$ for which $G$ contains a $k$-core.

# Algorithm for $k$-Core Existence



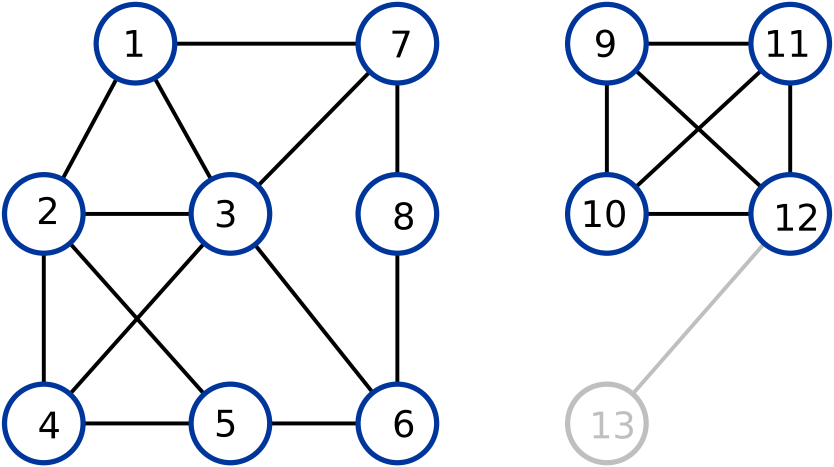

- Repeatedly delete all nodes of degree $< k$ until

# Algorithm for $k$-Core Existence



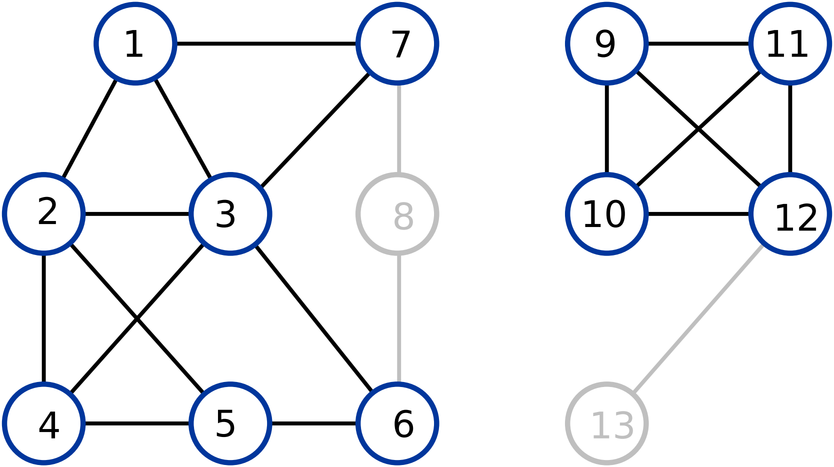

- Repeatedly delete all nodes of degree $< k$ until every remaining node has degree $\geq k$.
- Resulting graph is the largest $k$-core.

# Algorithm for $k$-Core Existence



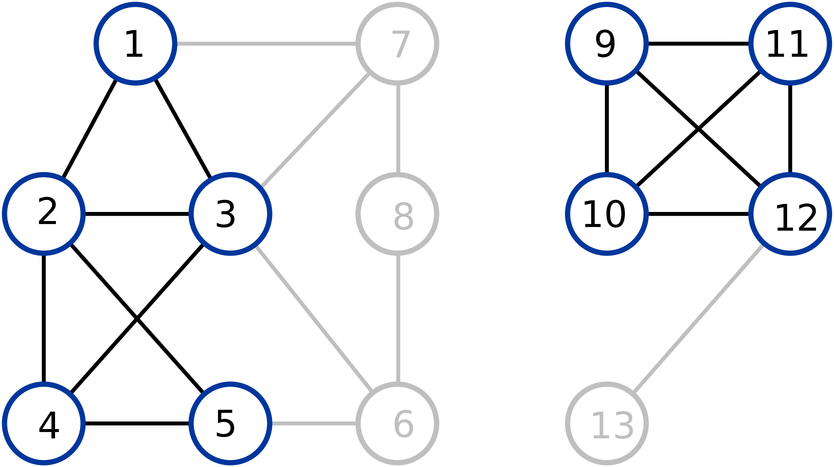

- Repeatedly delete all nodes of degree $< k$ until every remaining node has degree $\geq k$.
- Resulting graph is the largest $k$-core.

# Algorithm for $k$-Core Existence



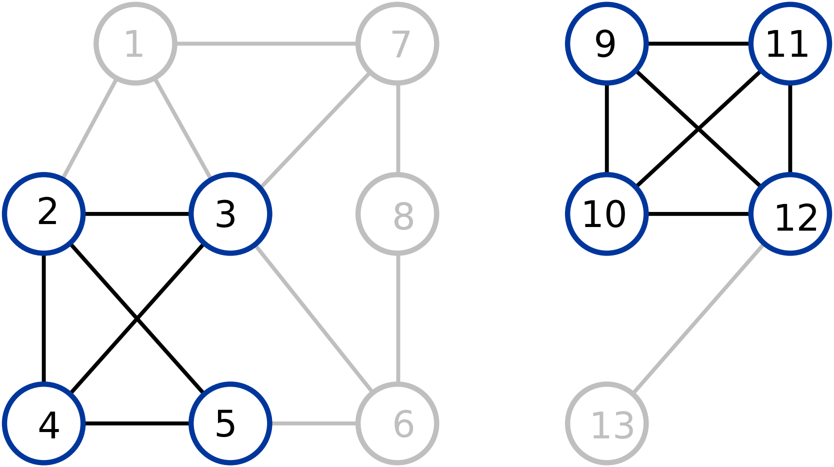

- Repeatedly delete all nodes of degree $< k$ until every remaining node has degree $\geq k$.
- Resulting graph is the largest $k$-core.

# Algorithm for $k$-Core Existence



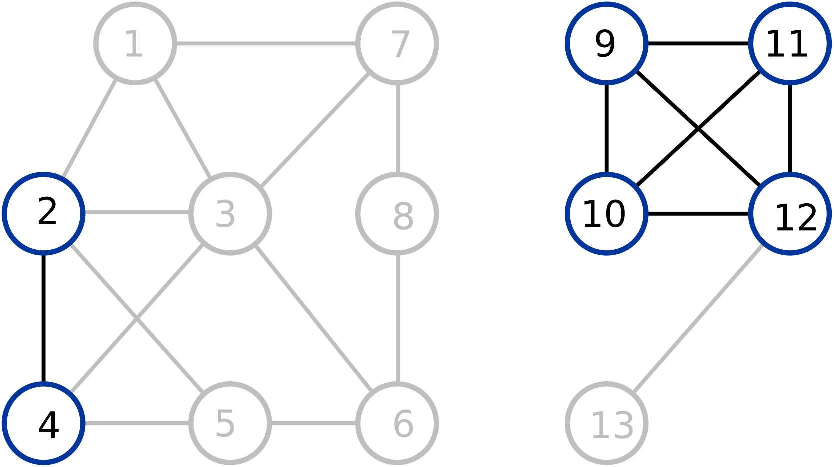

- Repeatedly delete all nodes of degree $< k$ until every remaining node has degree $\geq k$.
- Resulting graph is the largest $k$-core.

# Algorithm for $k$-Core Existence



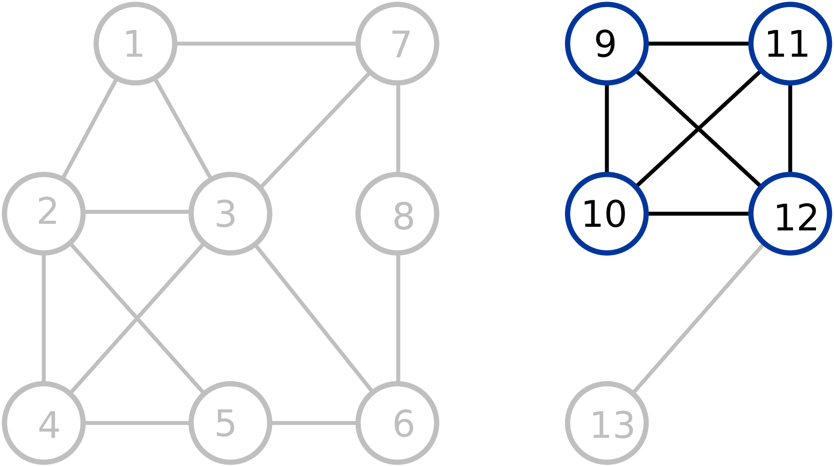

- Repeatedly delete all nodes of degree $< k$ until every remaining node has degree $\geq k$.
- Resulting graph is the largest $k$-core.

# Algorithm for $k$-Core Existence




- Repeatedly delete all nodes of degree $< k$ until every remaining node has degree $\geq k$.
- Resulting graph is the largest $k$-core.

# Algorithm for $k$-Core Existence




- Repeatedly delete all nodes of degree $< k$ until every remaining node has degree $\geq k$.
- Resulting graph is the largest $k$-core.

# Correctness of $k$-Core Existence Algorithm

- Repeatedly delete all nodes of degree $< k$ until every remaining node has degree $\geq k$.
- Why should the resulting graph $H$ be a $k$-core?
- Why should the resulting graph $H$ be the $k$-core with the largest number of nodes?

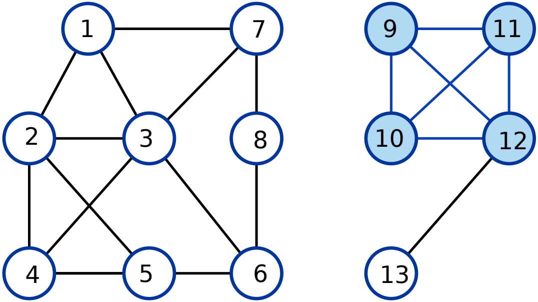
# Correctness of $k$-Core Existence Algorithm

- Repeatedly delete all nodes of degree $< k$ until every remaining node has degree $\geq k$.
- Why should the resulting graph $H$ be a $k$-core?
- Why should the resulting graph $H$ be the $k$-core with the largest number of nodes?
- Proof by contradiction.
  - Suppose there is a $k$-core $H'$ with more nodes than $H$.
  - Then $H \cup H'$ is also a $k$-core.
  - Moreover, no node in $H'$ will be deleted by the algorithm.

# Correctness of $k$-Core Existence Algorithm

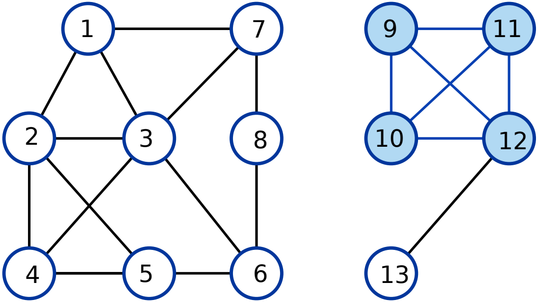
- Repeatedly delete all nodes of degree $< k$ until every remaining node has degree $\geq k$.
- Why should the resulting graph $H$ be a $k$-core?
- Why should the resulting graph $H$ be the $k$-core with the largest number of nodes?
- Proof by contradiction.
    - Suppose there is a $k$-core $H'$ with more nodes than $H$.
    - Then $H \cup H'$ is also a $k$-core.
    - Moreover, no node in $H'$ will be deleted by the algorithm.
- How do we implement $k$-core algorithm efficiently?

# Cores vs. Cliques



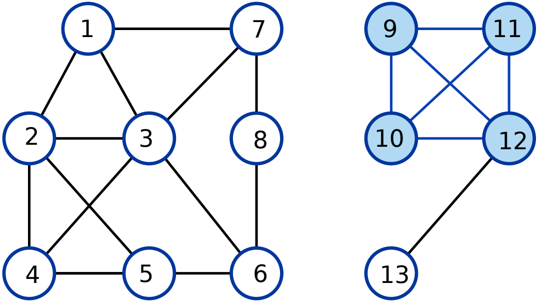

- A clique with $k$ nodes is a $(k-1)$-core.
- Can we use the $k$-core algorithm to find maximum cliques?

# Cores vs. Cliques



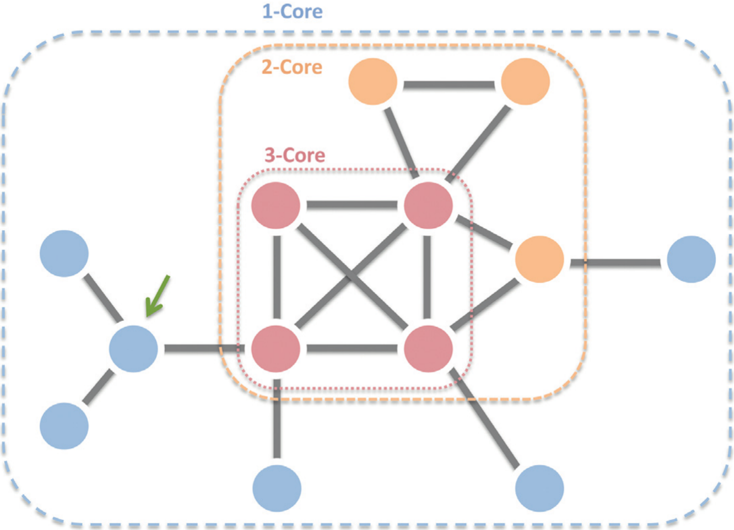

- A clique with $k$ nodes is a $(k-1)$-core.
- Can we use the $k$-core algorithm to find maximum cliques?
- Idea: Compute the largest value of $k$ for which a $k$-core $H$ exists. If $H$ is a clique, it must be the largest clique (of size $k+1$) in the graph.

# Cores vs. Cliques



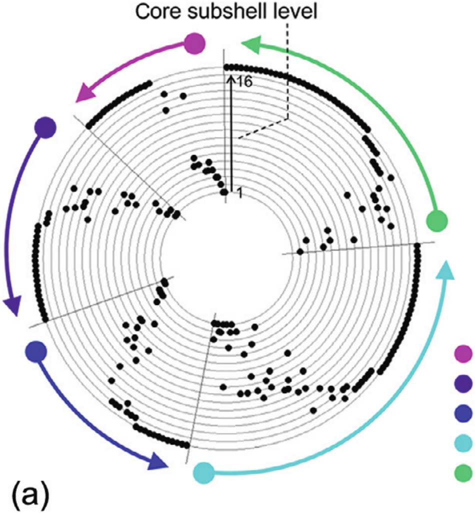

- A clique with $k$ nodes is a $(k-1)$-core.
- Can we use the $k$-core algorithm to find maximum cliques?
- Idea: Compute the largest value of $k$ for which a $k$-core $H$ exists. If $H$ is a clique, it must be the largest clique (of size $k+1$) in the graph.
- Flaw is that $H$ may not be a clique, in general. The largest clique may be disjoint from $H$ or be a subgraph of $H$.
- Moreover, the maximum clique may have $l$ nodes while there may be a $k$-core where $k > l - 1$, e.g., $k = 3$ and $l = 3$. Create such an example.

# *k*-Core Decomposition



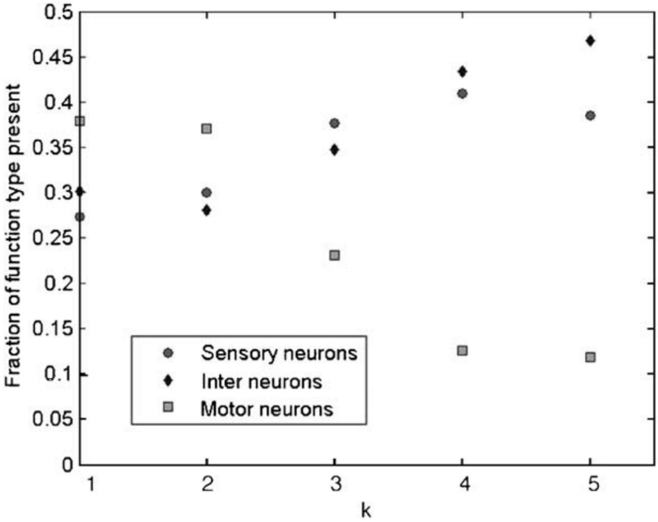

- Label each node by the *k*-core to which it belongs.

# *k*-Core Decomposition of Macaque Cortex



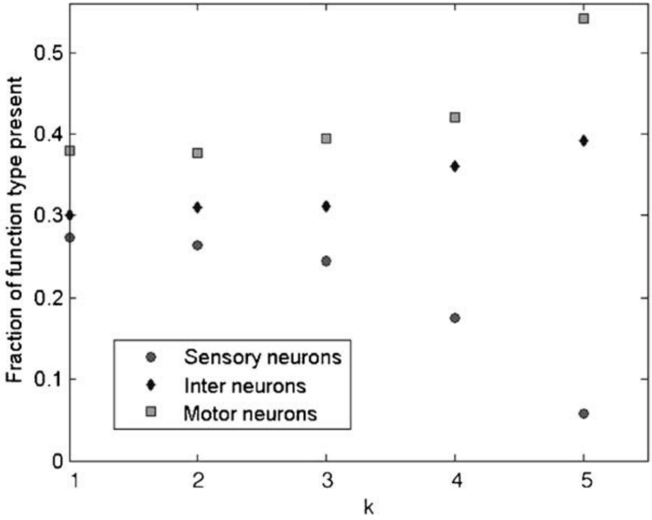

- 242-region macaque cortical connectome containing a 16-core.

# *k*-Core Decomposition of C. Elegans Connectome



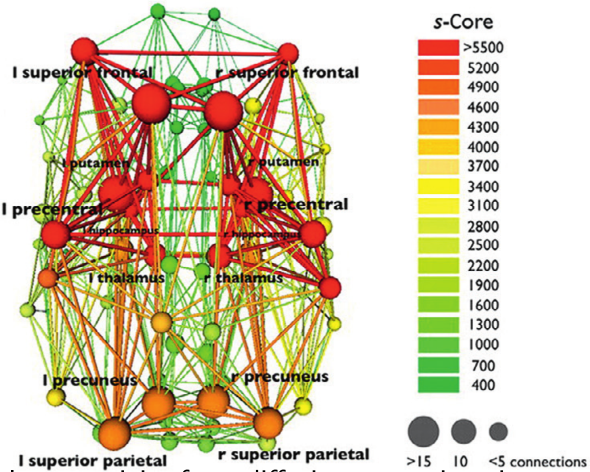

- Sensory neurons comprise the innermost cores based on out-degree.

# $k$-Core Decomposition of C. Elegans Connectome




- Sensory neurons comprise the innermost cores based on out-degree.
- Motor neurons comprise the inner-most cores based on in-degree.

# *s*-Core Decomposition of Human Connectome

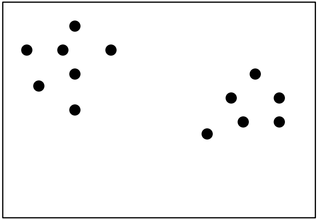


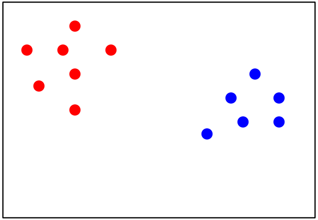

- Structural connectivity from diffusion tensor imaging.
- Connectome is the average of 21 individuals.
- Extend *k*-core algorithm to weighted networks.

# Modules and Clustering

- Finding modules or clusters formed by a set of objects is a widely studied problem.
- Long history in mathematics, statistics, and computer science.
- Module $\equiv$ Cluster $\equiv$ Community.

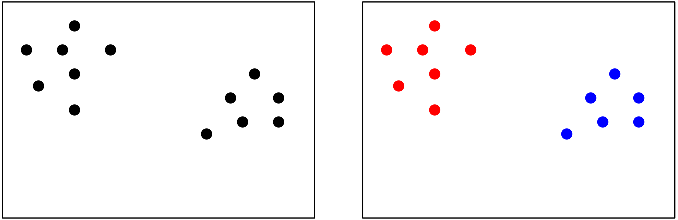
# Definition of Clustering





*Given a set of n objects, find the best partition of the objects into subsets such that each subset contains objects that are similar/close to each other.*

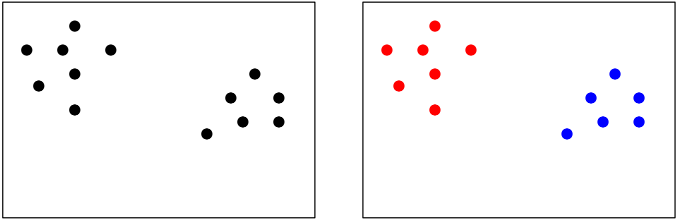
# Definition of Clustering



*Given a set of $n$ objects, find the best partition of the objects into subsets such that each subset contains objects that are similar/close to each other.*

- How do we measure how similar or close two objects are?

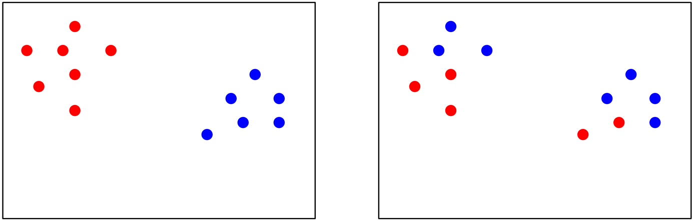
# Definition of Clustering



*Given a set of n objects, find the best partition of the objects into subsets such that each subset contains objects that are similar/close to each other.*

- How do we measure how similar or close two objects are?
- How many subsets?

# Definition of Clustering



*Given a set of n objects, find the best partition of the objects into subsets such that each subset contains objects that are similar/close to each other.*

- How do we measure how similar or close two objects are?
- How many subsets?
- How do we compare two different partitions?

# Measuring Similarity of Objects

- Assume each object specified by a list of values, e.g., $x$, $y$, $z$ coordinates indicating voxel position in an fMRI image.

# Measuring Similarity of Objects

- Assume each object specified by a list of values, e.g., $x$, $y$, $z$ coordinates indicating voxel position in an fMRI image.
- Distance between two objects $p$ and $q$ is $d(p, q)$.
- Euclidean metric: $d(p, q) = \sqrt{\sum_i (p_i - q_i)^2}$.

# Measuring Similarity of Objects

- Assume each object specified by a list of values, e.g., $x$, $y$, $z$ coordinates indicating voxel position in an fMRI image.
- Distance between two objects $p$ and $q$ is $d(p, q)$.
- Euclidean metric: $d(p, q) = \sqrt{\sum_i (p_i - q_i)^2}$.
- Manhattan metric: $d(p, q) = \sum_i |p_i - q_i|$.

# Measuring Similarity of Objects

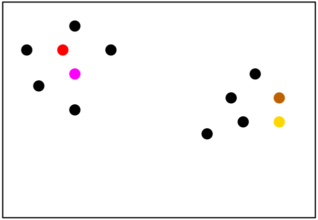
- Assume each object specified by a list of values, e.g., $x$, $y$, $z$ coordinates indicating voxel position in an fMRI image.
- Distance between two objects $p$ and $q$ is $d(p, q)$.
- Euclidean metric: $d(p, q) = \sqrt{\sum_i (p_i - q_i)^2}$.
- Manhattan metric: $d(p, q) = \sum_i |p_i - q_i|$.
- Other distances: normalised dot product, K-L divergence, relative entropy, Pearson's correlation.

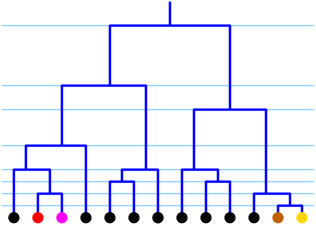
# Measuring Similarity of Objects

- Assume each object specified by a list of values, e.g., $x$, $y$, $z$ coordinates indicating voxel position in an fMRI image.
- Distance between two objects $p$ and $q$ is $d(p, q)$.
- Euclidean metric: $d(p, q) = \sqrt{\sum_i (p_i - q_i)^2}$.
- Manhattan metric: $d(p, q) = \sum_i |p_i - q_i|$.
- Other distances: normalised dot product, K-L divergence, relative entropy, Pearson's correlation.
- Metrics obey triangle inequality: $d(p, q) + d(q, r) \geq d(p, r)$.
  - Euclidean, Manhattan distances are metrics.
  - Correlation, dot product are not metrics.

# Hierarchical Clustering

- Attempt to recursively find sub-modules within modules.
- Natural way to "zoom into" areas of interest.
- Represent using a tree or dendrogram.





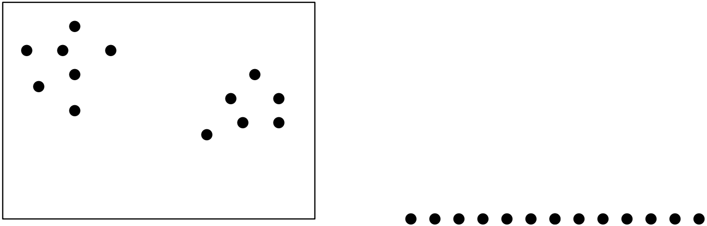
# Hierarchical Clustering Algorithm

- Bottom-up clustering algorithm.

# Hierarchical Clustering Algorithm

- Bottom-up clustering algorithm.

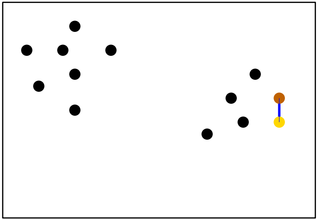
1. Start with every object in its own cluster.



# Hierarchical Clustering Algorithm

- Bottom-up clustering algorithm.

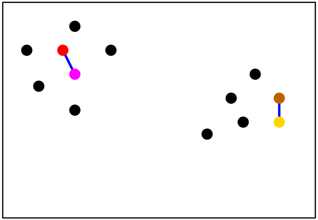
1. Start with every object in its own cluster.
2. Repeat
   - Let $C_i$ and $C_j$ be the clusters "nearest" each other.
   - Merge $C_i$ and $C_j$.



# Hierarchical Clustering Algorithm

- Bottom-up clustering algorithm.

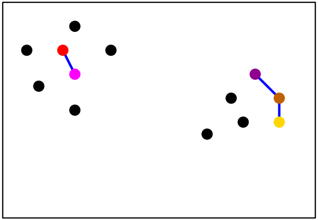
1. Start with every object in its own cluster.
2. Repeat
   - Let $C_i$ and $C_j$ be the clusters "nearest" each other.
   - Merge $C_i$ and $C_j$.



# Hierarchical Clustering Algorithm

- Bottom-up clustering algorithm.

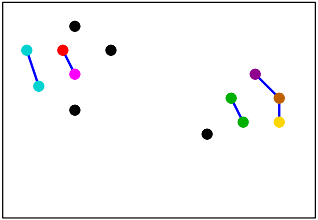
1. Start with every object in its own cluster.
2. Repeat
   - Let $C_i$ and $C_j$ be the clusters "nearest" each other.
   - Merge $C_i$ and $C_j$.



# Hierarchical Clustering Algorithm

- Bottom-up clustering algorithm.

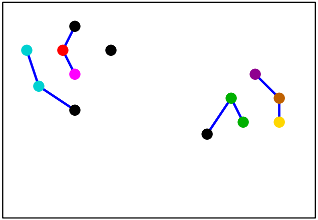
1. Start with every object in its own cluster.
2. Repeat
   - Let $C_i$ and $C_j$ be the clusters "nearest" each other.
   - Merge $C_i$ and $C_j$.

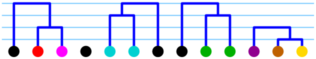


# Hierarchical Clustering Algorithm

- Bottom-up clustering algorithm.

1. Start with every object in its own cluster.
2. Repeat
   - Let $C_i$ and $C_j$ be the clusters "nearest" each other.
   - Merge $C_i$ and $C_j$.

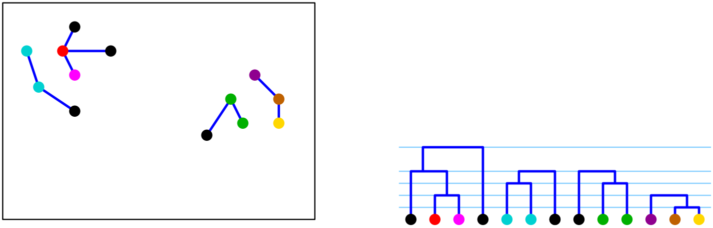




# Hierarchical Clustering Algorithm

- Bottom-up clustering algorithm.

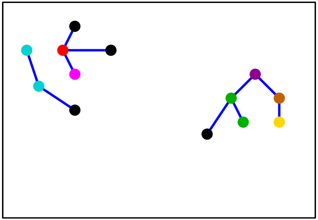
1. Start with every object in its own cluster.
2. Repeat
   - Let $C_i$ and $C_j$ be the clusters "nearest" each other.
   - Merge $C_i$ and $C_j$.

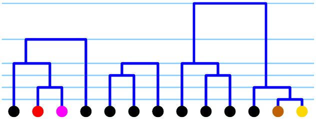


# Hierarchical Clustering Algorithm

- Bottom-up clustering algorithm.

1. Start with every object in its own cluster.
2. Repeat
   - Let $C_i$ and $C_j$ be the clusters "nearest" each other.
   - Merge $C_i$ and $C_j$.

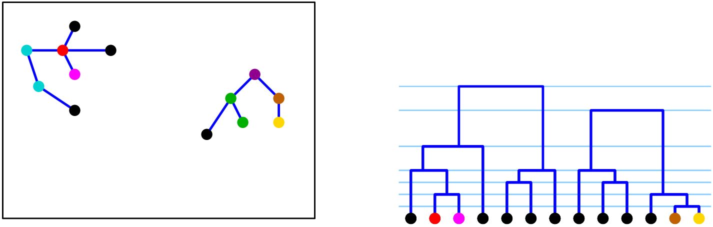




# Hierarchical Clustering Algorithm

- Bottom-up clustering algorithm.

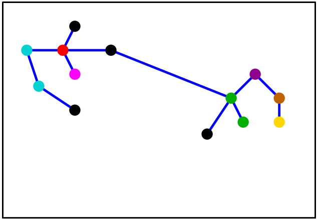
1. Start with every object in its own cluster.
2. Repeat
   - Let $C_i$ and $C_j$ be the clusters "nearest" each other.
   - Merge $C_i$ and $C_j$.

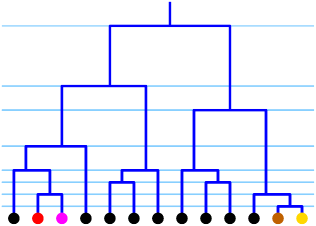


# Hierarchical Clustering Algorithm

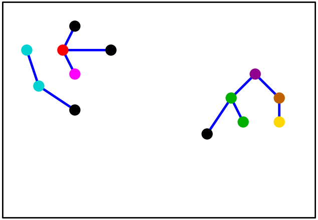
- Bottom-up clustering algorithm.

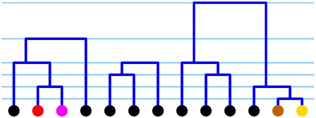
1. Start with every object in its own cluster.
2. Repeat
   - Let $C_i$ and $C_j$ be the clusters "nearest" each other.
   - Merge $C_i$ and $C_j$.
3. until all the objects are in one cluster.





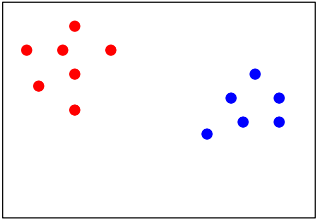
# Measuring Distance between Clusters





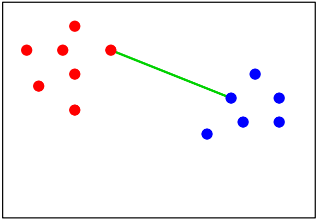
- How do we measure distance between two clusters $C_i$ and $C_j$?

# Measuring Distance between Clusters



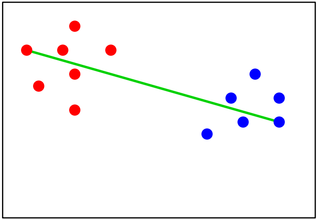
- How do we measure distance between two clusters $C_i$ and $C_j$?

# Measuring Distance between Clusters



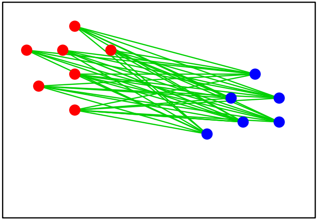
- How do we measure distance between two clusters $C_i$ and $C_j$?
- $d_{min}(C_i, C_j) =$ distance between closest pair of objects.

# Measuring Distance between Clusters



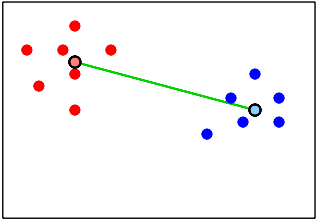
- How do we measure distance between two clusters $C_i$ and $C_j$?
- $d_{min}(C_i, C_j) =$ distance between closest pair of objects.
- $d_{max}(C_i, C_j) =$ distance between farthest pair of objects.

# Measuring Distance between Clusters



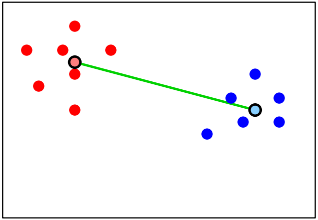
- How do we measure distance between two clusters $C_i$ and $C_j$?
- $d_{min}(C_i, C_j) =$ distance between closest pair of objects.
- $d_{max}(C_i, C_j) =$ distance between farthest pair of objects.
- $d_{mean}(C_i, C_j) =$ average of distances between all pairs of objects.

# Measuring Distance between Clusters



- How do we measure distance between two clusters $C_i$ and $C_j$?
- $d_{min}(C_i, C_j) =$ distance between closest pair of objects.
- $d_{max}(C_i, C_j) =$ distance between farthest pair of objects.
- $d_{mean}(C_i, C_j) =$ average of distances between all pairs of objects.
- $d_{centroid}(C_i, C_j) = d(\mu_i, \mu_j)$, where $\mu_i$ is the centroid of $C_i$.

# Measuring Distance between Clusters



- How do we measure distance between two clusters $C_i$ and $C_j$?
- $d_{min}(C_i, C_j) =$ distance between closest pair of objects.
- $d_{max}(C_i, C_j) =$ distance between farthest pair of objects.
- $d_{mean}(C_i, C_j) =$ average of distances between all pairs of objects.
- $d_{centroid}(C_i, C_j) = d(\mu_i, \mu_j)$, where $\mu_i$ is the centroid of $C_i$.
- Methods are called minimum linkage, maximum linkage, mean linkage, and centroid linkage clustering, respectively.
- Computing $d_{min}, d_{max}, d_{avg}$ takes $O(n_i n_j)$ time.
- Computing $d_{mean}$ takes $O(n_i + n_j)$ time.

# Running Time of Hierarchical Clustering

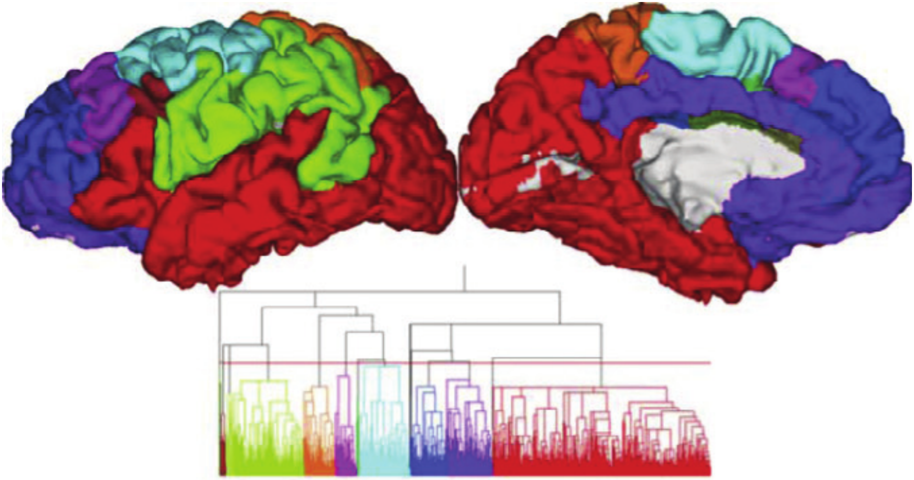
1. Start with every object in its own cluster.
2. Repeat
   - Let $D_i$ and $D_j$ be the clusters "nearest" each other.
   - Merge $D_i$ and $D_j$.
3. until all the objects are in one cluster.

# Running Time of Hierarchical Clustering

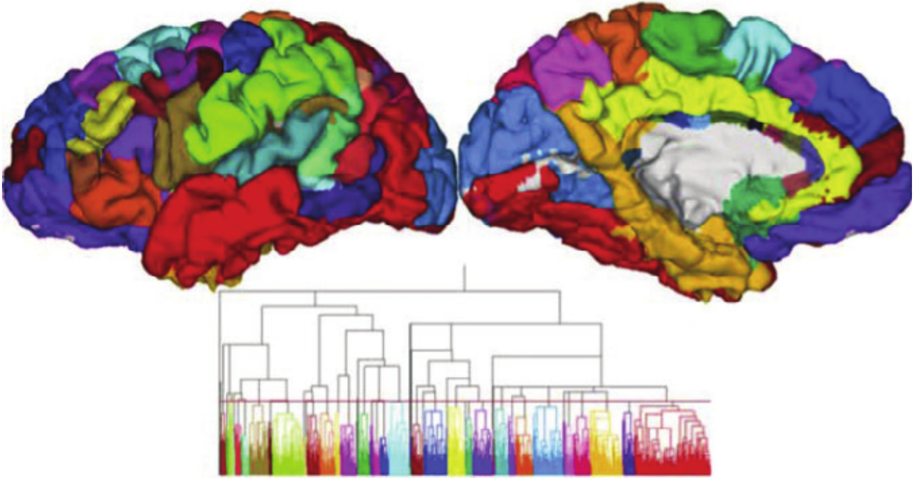
1. Start with every object in its own cluster.
2. Repeat
   - Let $D_i$ and $D_j$ be the clusters "nearest" each other.
   - Merge $D_i$ and $D_j$.
3. until all the objects are in one cluster.

- Assume computing distance between two objects takes $O(1)$ time.
- Store all $O(n^2)$ inter-object distances.
- At each iteration, compute distance between every pair of clusters: takes $O(n^2)$ time in total.
- There are $n$ iterations, so overall running time is $O(nn^2) = O(n^3)$.

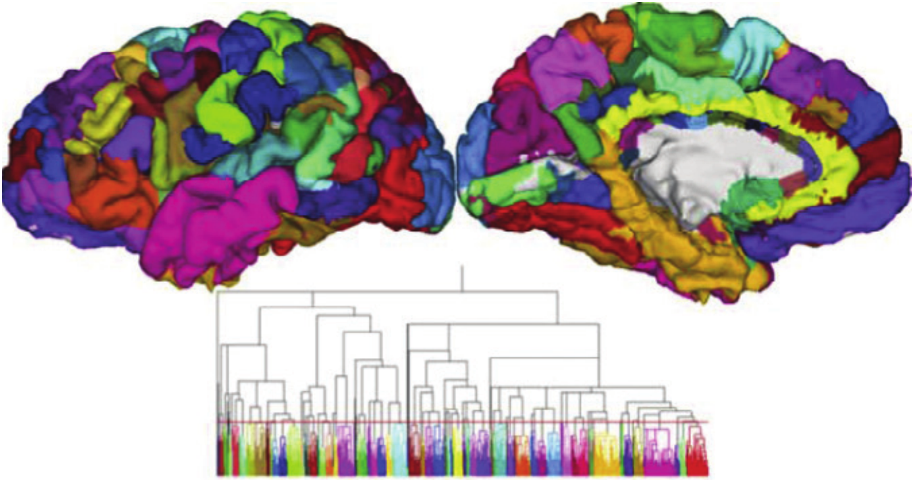
# Hierarchical Clustering Result



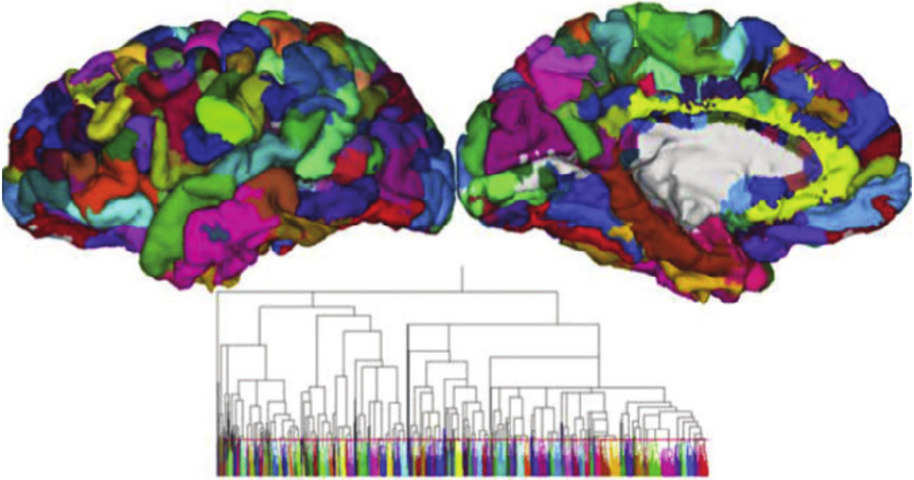
# Hierarchical Clustering Result



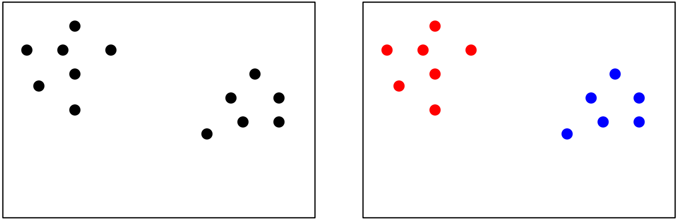
# Hierarchical Clustering Result



# Hierarchical Clustering Result



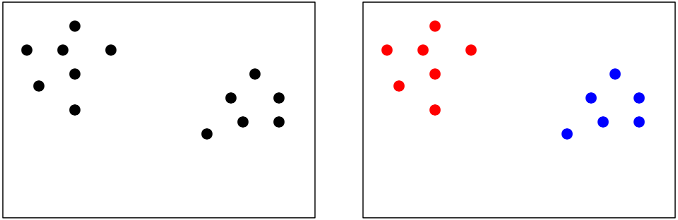
# Definition of Clustering



*Given a set of $n$ objects, find the best partition of the objects into subsets such that each subset contains objects that are similar/close to each other.*

- How do we measure how similar or close two objects are?

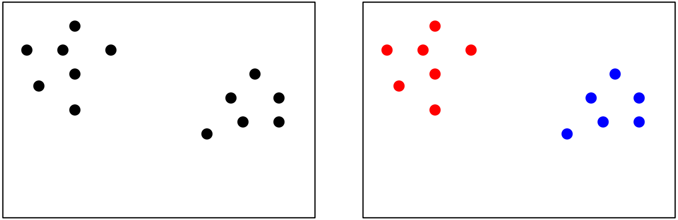
# Definition of Clustering



*Given a set of n objects, find the best partition of the objects into subsets such that each subset contains objects that are similar/close to each other.*

- How do we measure how similar or close two objects are?
- How many subsets?

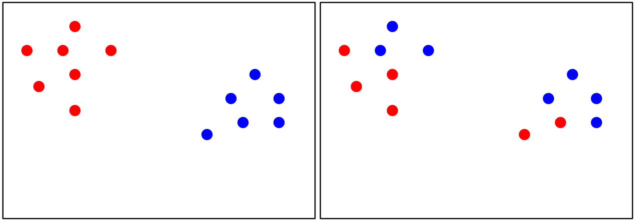
# Definition of Clustering



*Given a set of n objects, find the best partition of the objects into subsets such that each subset contains objects that are similar/close to each other.*

- How do we measure how similar or close two objects are?
- How many subsets? Not specified in hierarchical clustering.

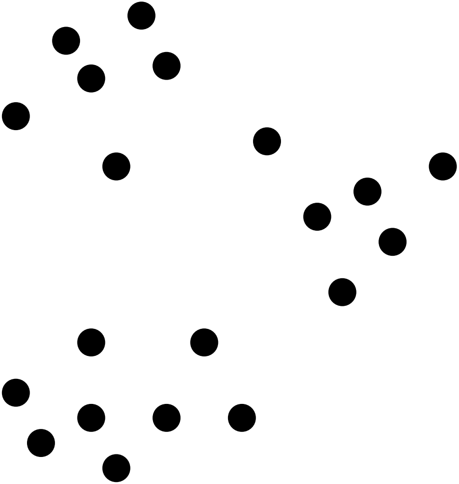
# Definition of Clustering



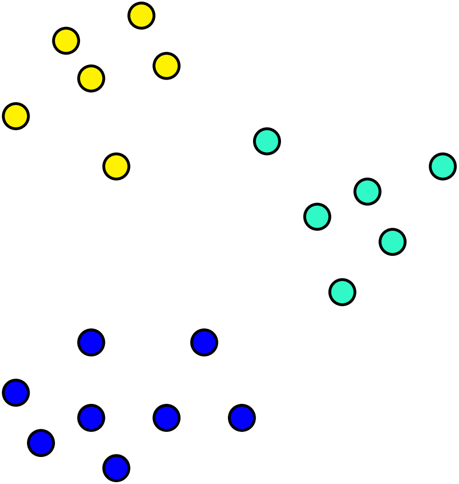
*Given a set of n objects, find the best partition of the objects into subsets such that each subset contains objects that are similar/close to each other.*

- How do we measure how similar or close two objects are?
- How many subsets? Not specified in hierarchical clustering.
- How do we compare two different partitions?

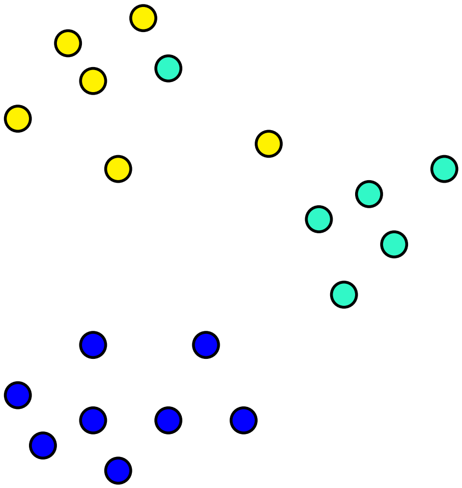
# Example of Clustering



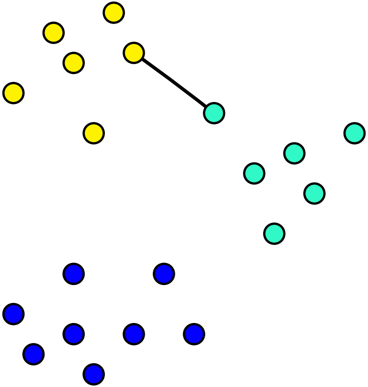
# Example of Clustering

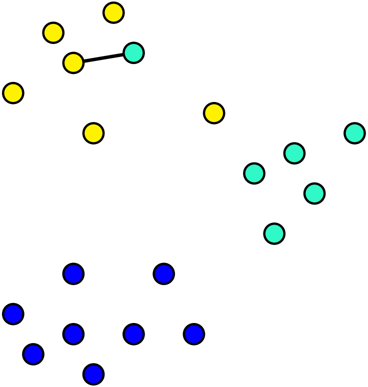


# Example of Clustering



# Example of Clustering





# Formalising the Clustering Problem

- Let $U$ be the set of $n$ objects labelled $p_1, p_2, \ldots, p_n$.
- For every pair $p_i$ and $p_j$, we have a distance $d(p_i, p_j)$.
- We require $d(p_i, p_i) = 0$, $d(p_i, p_j) > 0$, if $i \neq j$, and $d(p_i, p_j) = d(p_j, p_i)$

# Formalising the Clustering Problem

- Let $U$ be the set of $n$ objects labelled $p_1, p_2, \ldots, p_n$.
- For every pair $p_i$ and $p_j$, we have a distance $d(p_i, p_j)$.
- We require $d(p_i, p_i) = 0$, $d(p_i, p_j) > 0$, if $i \neq j$, and $d(p_i, p_j) = d(p_j, p_i)$
- Given a positive integer $k$, a *k-clustering* of $U$ is a partition of $U$ into $k$ non-empty subsets or "clusters" $C_1, C_2, \ldots C_k$.

# Formalising the Clustering Problem

- Let $U$ be the set of $n$ objects labelled $p_1, p_2, \ldots, p_n$.
- For every pair $p_i$ and $p_j$, we have a distance $d(p_i, p_j)$.
- We require $d(p_i, p_i) = 0$, $d(p_i, p_j) > 0$, if $i \neq j$, and $d(p_i, p_j) = d(p_j, p_i)$
- Given a positive integer $k$, a *k-clustering* of $U$ is a partition of $U$ into $k$ non-empty subsets or "clusters" $C_1, C_2, \ldots C_k$.
- The *spacing* of a clustering is the smallest distance between objects in two different subsets:

$$\text{spacing}(C_1, C_2, \ldots C_k) = \min_{\substack{1 \leq i,j \leq k \\ i \neq j, \\ p \in C_i, q \in C_j}} d(p, q)$$

# Formalising the Clustering Problem

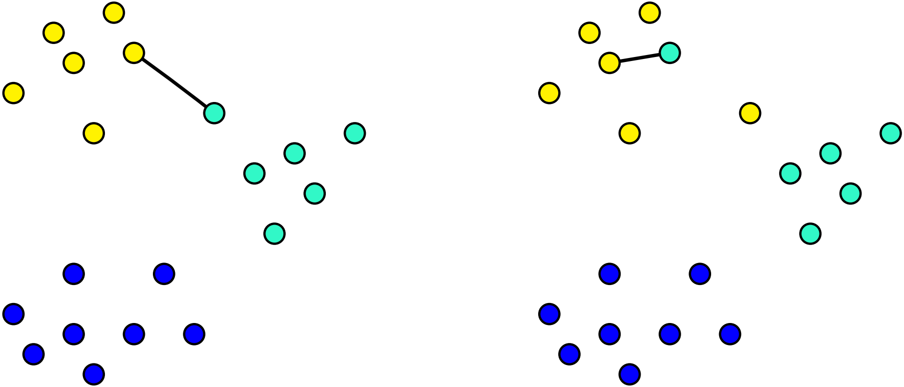
- Let $U$ be the set of $n$ objects labelled $p_1, p_2, \ldots, p_n$.
- For every pair $p_i$ and $p_j$, we have a distance $d(p_i, p_j)$.
- We require $d(p_i, p_i) = 0$, $d(p_i, p_j) > 0$, if $i \neq j$, and $d(p_i, p_j) = d(p_j, p_i)$
- Given a positive integer $k$, a *k-clustering* of $U$ is a partition of $U$ into $k$ non-empty subsets or "clusters" $C_1, C_2, \ldots C_k$.
- The *spacing* of a clustering is the smallest distance between objects in two different subsets:

$$\text{spacing}(C_1, C_2, \ldots C_k) = \min_{\substack{1 \leq i,j \leq k \\ i \neq j, \\ p \in C_i, q \in C_j}} d(p, q)$$

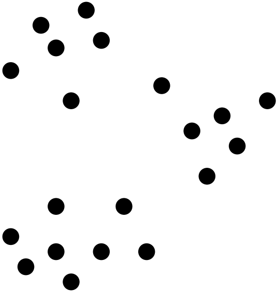
CLUSTERING OF MAXIMUM SPACING

Given a set $U$ of objects, a distance function $d : U \times U \rightarrow \mathbb{R}^+$,
and a positive integer $k$,
compute a $k$-clustering of $U$ whose spacing is the largest over all possible $k$-clusterings.

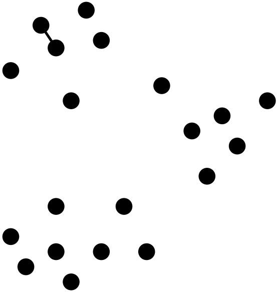
# Example of Clustering of Maximum Spacing



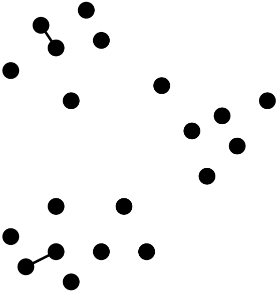
# Algorithm for Clustering of Maximum Spacing



# Algorithm for Clustering of Maximum Spacing

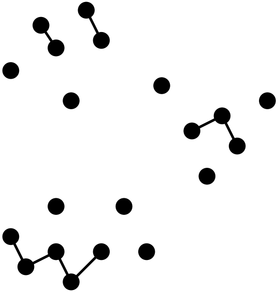


# Algorithm for Clustering of Maximum Spacing



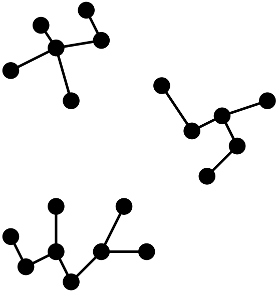
- Intuition: greedily cluster objects in increasing order of distance.

# Algorithm for Clustering of Maximum Spacing



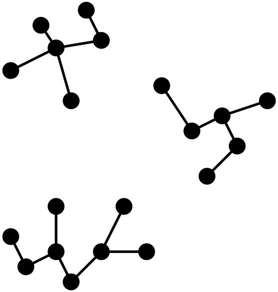
- Intuition: greedily cluster objects in increasing order of distance.

# Algorithm for Clustering of Maximum Spacing



- Intuition: greedily cluster objects in increasing order of distance.
- Let $\mathcal{C}$ be a set of $n$ clusters, with each object in $U$ in its own cluster.
- Process pairs of objects in increasing order of distance.
  - Let $(p, q)$ be the next pair with $p \in C_p$ and $q \in C_q$.
  - If $C_p \neq C_q$, add new cluster $C_p \cup C_q$ to $\mathcal{C}$, delete $C_p$ and $C_q$ from $\mathcal{C}$.
- Stop when there are $k$ clusters in $\mathcal{C}$.

# Algorithm for Clustering of Maximum Spacing

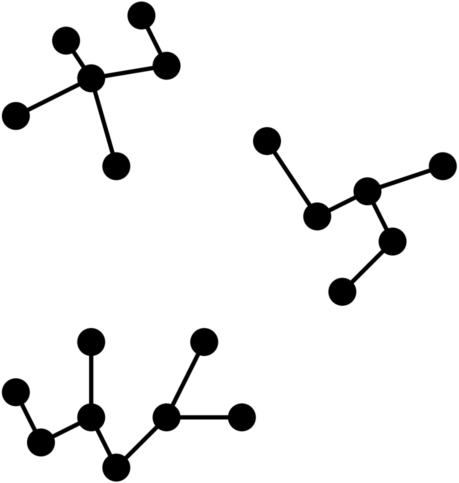


- Intuition: greedily cluster objects in increasing order of distance.
- Let $\mathcal{C}$ be a set of $n$ clusters, with each object in $U$ in its own cluster.
- Process pairs of objects in increasing order of distance.
  - Let $(p, q)$ be the next pair with $p \in C_p$ and $q \in C_q$.
  - If $C_p \neq C_q$, add new cluster $C_p \cup C_q$ to $\mathcal{C}$, delete $C_p$ and $C_q$ from $\mathcal{C}$.
- Stop when there are $k$ clusters in $\mathcal{C}$.
- Same as Kruskal's algorithm but do not add last $k - 1$ edges in MST.

Skip proof

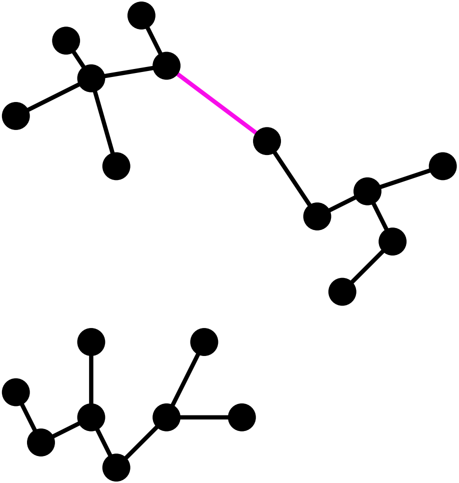
# What is the spacing of the Algorithm's Clustering?

- Let $\mathcal{C}$ be the clustering produced by the algorithm.
- What is spacing($\mathcal{C}$)?

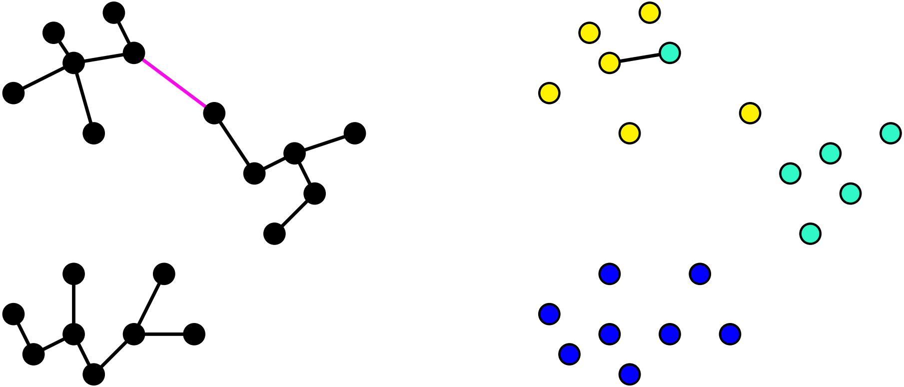


# What is the spacing of the Algorithm's Clustering?

- Let $\mathcal{C}$ be the clustering produced by the algorithm.
- What is spacing($\mathcal{C}$)? It is the cost of the $(k-1)$st most expensive edge in the MST. Let this cost be $d^*$.

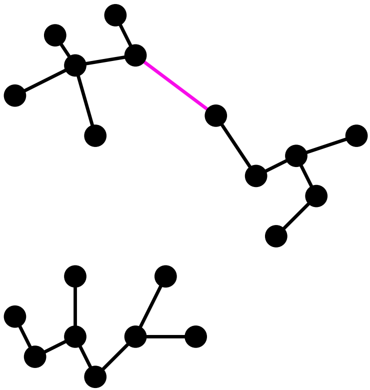


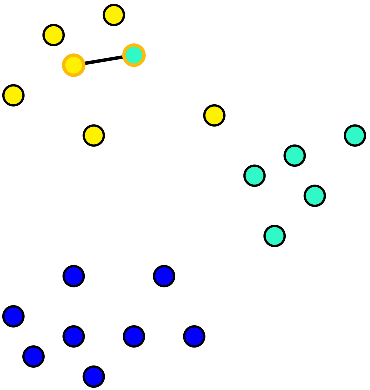
# Why does the Algorithm Work?



- Let $\mathcal{C}'$ be any other clustering.
- We will prove that spacing$(\mathcal{C}') \leq d^*$.

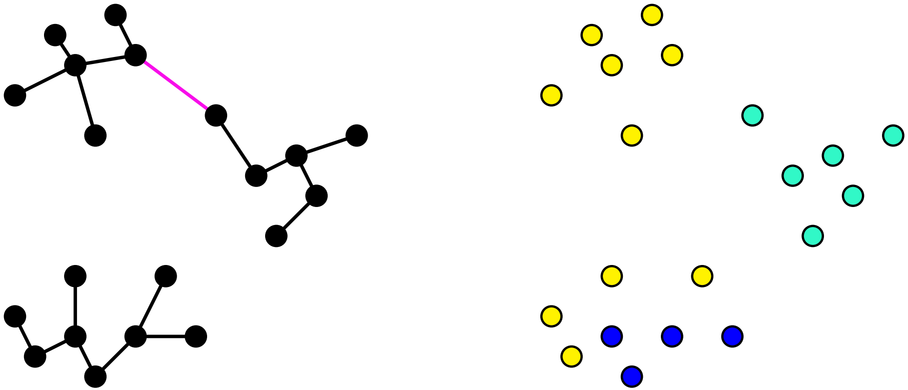
# Why does the Algorithm Work?



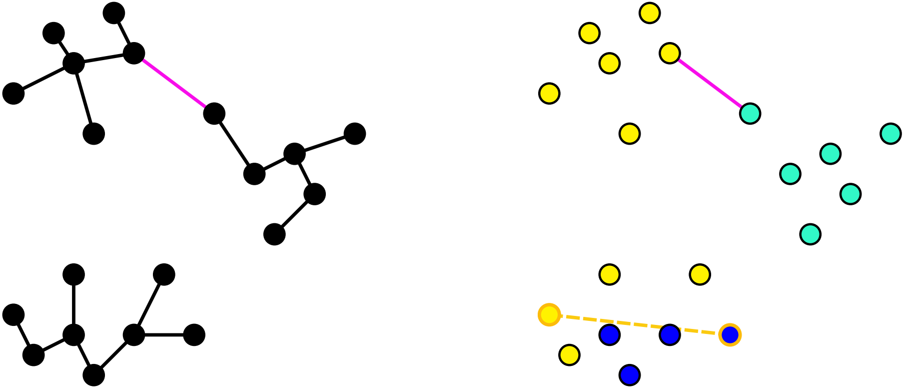


- Let $\mathcal{C}'$ be any other clustering.
- We will prove that spacing$(\mathcal{C}') \leq d^*$.

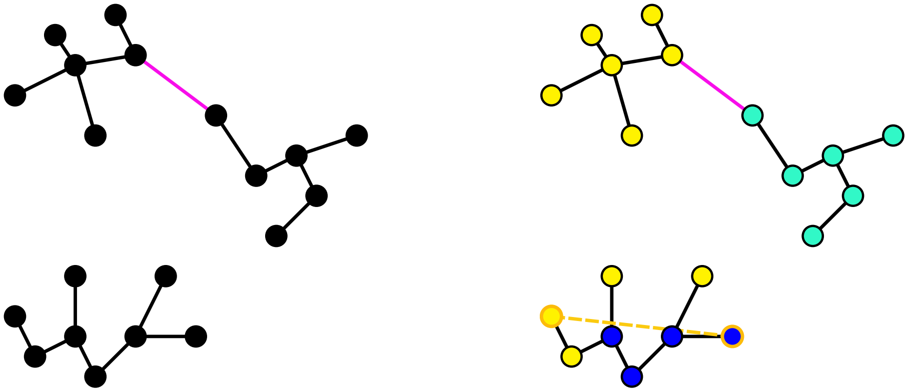
# spacing($\mathcal{C}'$) $\leq d^*$: Intuition



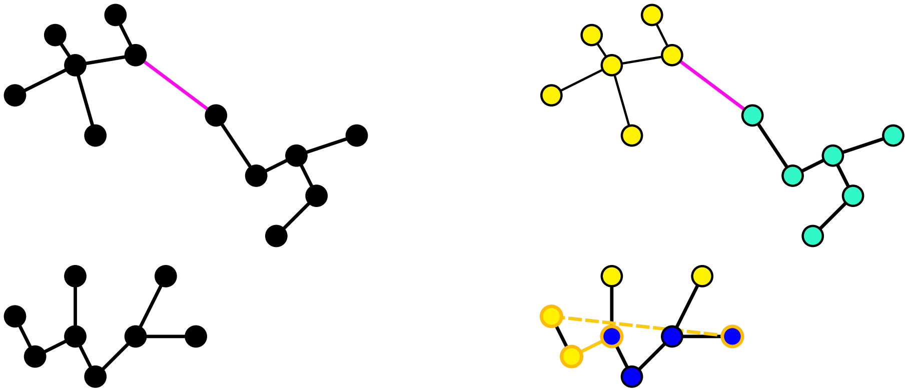
# spacing$(\mathcal{C}') \leq d^*$: Intuition



# spacing$(\mathcal{C}') \leq d^*$: Intuition



# spacing($\mathcal{C}'$) $\leq d^*$: Intuition



# spacing($\mathcal{C}'$) $\leq d^*$

- There must be two objects $p_i$ and $p_j$ in $U$ in the same cluster $C_r$ in $\mathcal{C}$ but in different clusters in $\mathcal{C}'$:

# spacing($\mathcal{C}'$) $\leq d^*$

- There must be two objects $p_i$ and $p_j$ in $U$ in the same cluster $C_r$ in $\mathcal{C}$ but in different clusters in $\mathcal{C}'$: spacing($\mathcal{C}'$) $\leq d(p_i, p_j)$.

# spacing$(\mathcal{C}') \leq d^*$

- There must be two objects $p_i$ and $p_j$ in $U$ in the same cluster $C_r$ in $\mathcal{C}$ but in different clusters in $\mathcal{C}'$: spacing$(\mathcal{C}') \leq d(p_i, p_j)$. But $d(p_i, p_j)$ could be $> d^*$.
- Suppose $p_i \in C'_s$ and $p_j \in C'_t$ in $\mathcal{C}'$.

# spacing($\mathcal{C}'$) $\leq d^*$

- There must be two objects $p_i$ and $p_j$ in $U$ in the same cluster $C_r$ in $\mathcal{C}$ but in different clusters in $\mathcal{C}'$: spacing($\mathcal{C}'$) $\leq d(p_i, p_j)$. But $d(p_i, p_j)$ could be $> d^*$.
- Suppose $p_i \in C'_s$ and $p_j \in C'_t$ in $\mathcal{C}'$.
- All edges in the path $Q$ connecting $p_i$ and $p_j$ in the MST have length $\leq d^*$.
- In particular, there is an object $p \in C'_s$ and an object $p' \notin C'_s$ such that $p$ and $p'$ are adjacent in $Q$.
- $d(p, p') \leq d^* \Rightarrow$ spacing($\mathcal{C}'$) $\leq d(p, p') \leq d^*$.

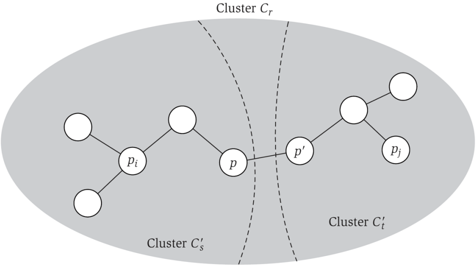
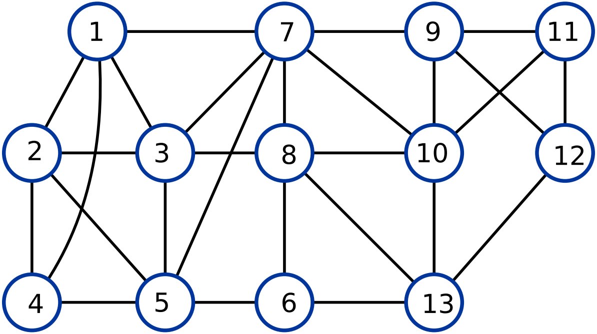



**Figure 4.15** An illustration of the proof of (4.26), showing that the spacing of any other clustering can be no larger than that of the clustering found by the single-linkage algorithm.

# Disadvantages of Hierarchical Clustering

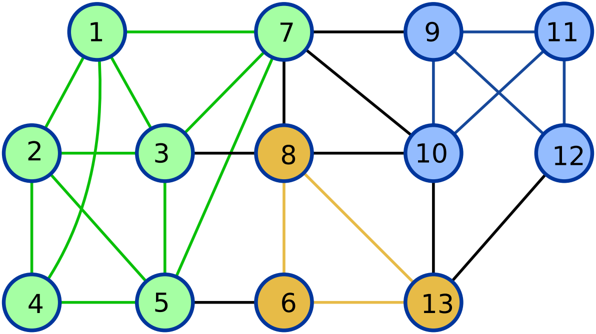
- To get a set of modules, at which level do we cut the dendrogram?
- Optimality due to spacing argument applies only to single linkage clustering.
- We need a different definition of module quality that captures connectivity within and across modules.

# Motivation



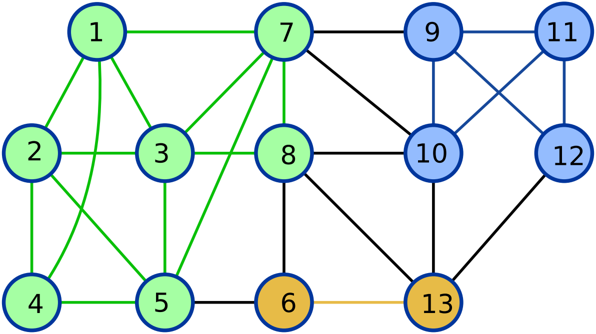

- Given an undirected, unweighted graph $G = (V, E)$ suppose we partition the nodes into $k$ modules $\mathcal{C} = C_1, C_2, \ldots C_k$.
- How do we measure the "quality" of $\mathcal{C}$?
- Intuition: many more edges within modules than among modules.

# Motivation



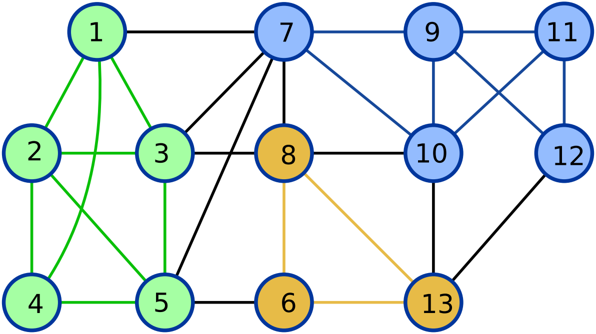

- Given an undirected, unweighted graph $G = (V, E)$ suppose we partition the nodes into $k$ modules $\mathcal{C} = C_1, C_2, \ldots C_k$.
- How do we measure the "quality" of $\mathcal{C}$?
- Intuition: many more edges within modules than among modules.

# Motivation



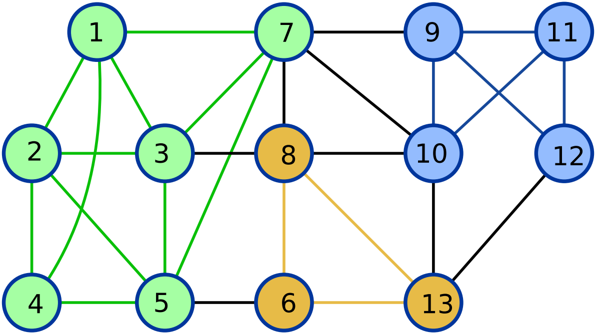

- Given an undirected, unweighted graph $G = (V, E)$ suppose we partition the nodes into $k$ modules $\mathcal{C} = C_1, C_2, \ldots C_k$.
- How do we measure the "quality" of $\mathcal{C}$?
- Intuition: many more edges within modules than among modules.

# Motivation



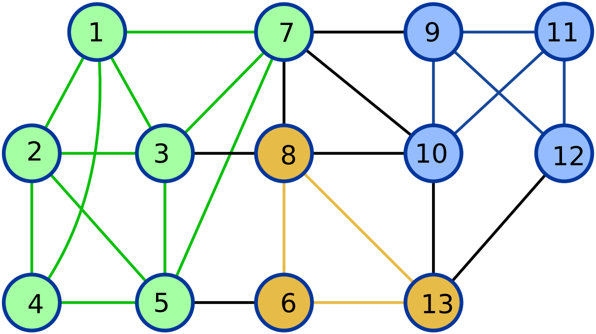

- Given an undirected, unweighted graph $G = (V, E)$ suppose we partition the nodes into $k$ modules $\mathcal{C} = C_1, C_2, \ldots C_k$.
- How do we measure the "quality" of $\mathcal{C}$?
- Intuition: many more edges within modules than among modules.

# Initial Definition of Modularity




- How do we count the number of edges within modules?

# Initial Definition of Modularity

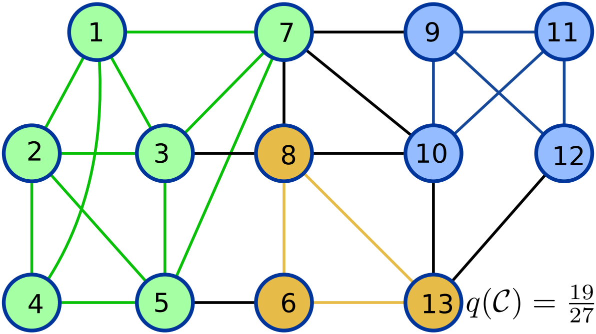



- How do we count the number of edges within modules?
- For every node $u \in V$, define $c(u)$ as the index of $u$'s module.

$$q(\mathcal{C}) = \frac{1}{m} \sum_{(u,v) \in E} \delta(c(u), c(v)), \text{ where } \delta \text{ is the Kronecker delta function}$$

$$= \frac{1}{2m} \sum_{u,v \in V} a(u,v)\delta(c(u), c(v)), \text{ where } a(u,v) = 1 \text{ iff } (u,v) \text{ is an edge}$$

# Initial Definition of Modularity

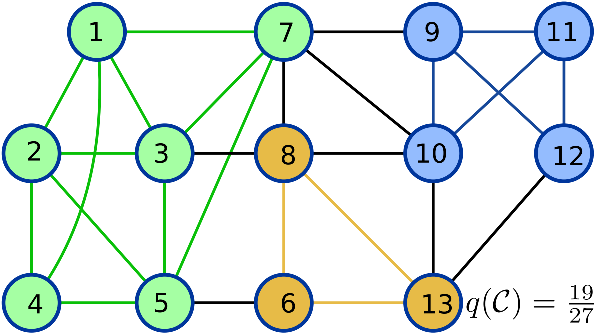



- How do we count the number of edges within modules?
- For every node $u \in V$, define $c(u)$ as the index of $u$'s module.

$$q(\mathcal{C}) = \frac{1}{m} \sum_{(u,v) \in E} \delta(c(u), c(v)), \text{ where } \delta \text{ is the Kronecker delta function}$$

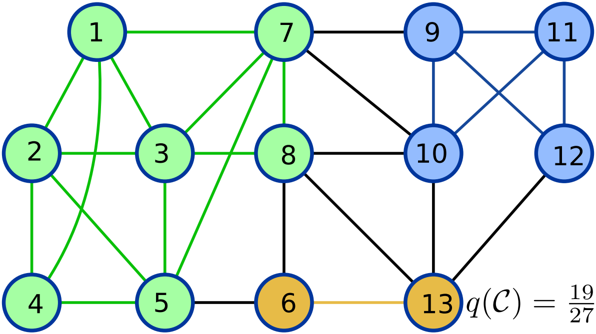
$$= \frac{1}{2m} \sum_{u,v \in V} a(u,v)\delta(c(u), c(v)), \text{ where } a(u,v) = 1 \text{ iff } (u,v) \text{ is an edge}$$

# Optimising Modularity




$$q(\mathcal{C}) = \frac{1}{2m} \sum_{u,v \in V} a(u,v)\delta(c(u), c(v))$$

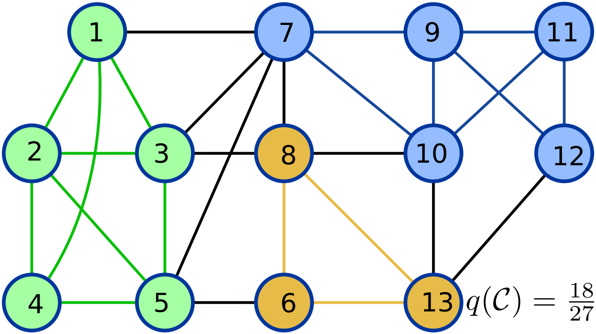
# Optimising Modularity




$q(\mathcal{C}) = \frac{19}{27}$

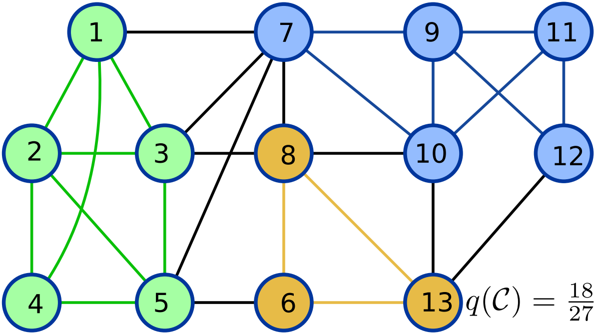
$$q(\mathcal{C}) = \frac{1}{2m} \sum_{u,v \in V} a(u,v)\delta(c(u), c(v))$$

# Optimising Modularity




$$q(\mathcal{C}) = \frac{1}{2m} \sum_{u,v \in V} a(u,v)\delta(c(u), c(v))$$

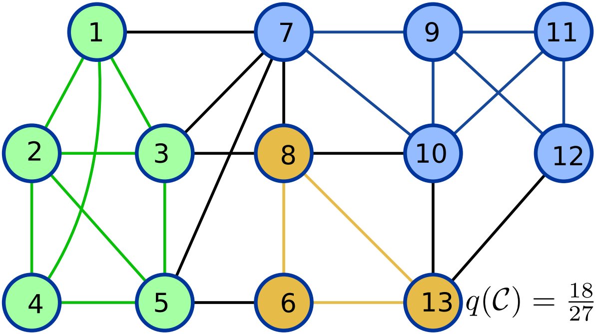
# Optimising Modularity




$$q(\mathcal{C}) = \frac{1}{2m} \sum_{u,v \in V} a(u,v)\delta(c(u), c(v))$$

- Should we maximise or minimise $q(\mathcal{C})$?

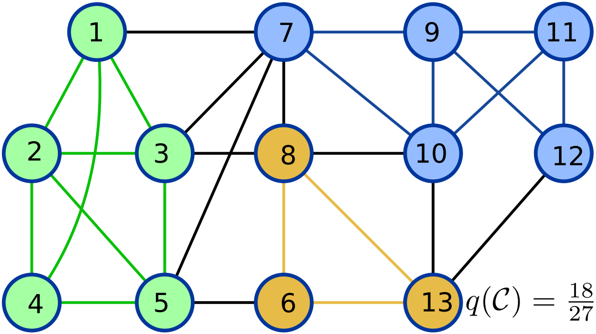
# Optimising Modularity




$$q(\mathcal{C}) = \frac{1}{2m} \sum_{u,v \in V} a(u,v)\delta(c(u), c(v))$$

- Should we maximise or minimise $q(\mathcal{C})$? Maximise it.
- What is the value of $q(\mathcal{C})$ if we place all nodes in $G$ in a single cluster?

# Optimising Modularity



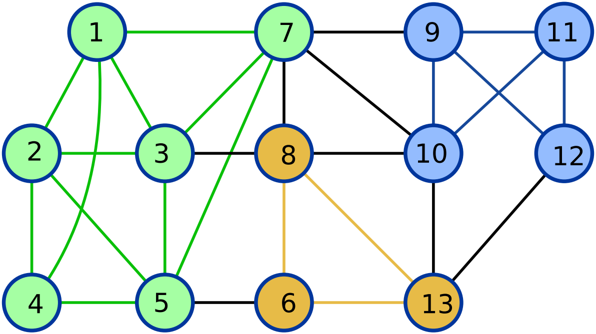

$$q(\mathcal{C}) = \frac{1}{2m} \sum_{u,v \in V} a(u,v)\delta(c(u), c(v))$$

- Should we maximise or minimise $q(\mathcal{C})$? Maximise it.
- What is the value of $q(\mathcal{C})$ if we place all nodes in $G$ in a single cluster? 1!

# Two Criteria for High Quality Partitions

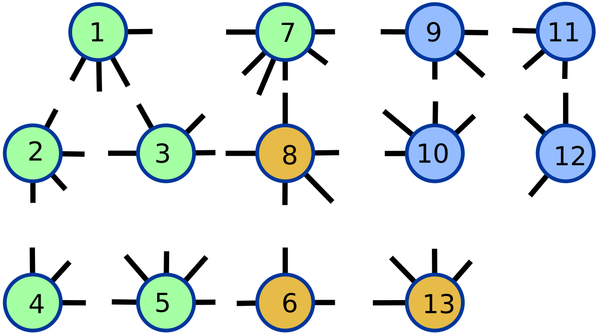
1. Nodes are in highly cohesive modules, i.e., nodes within the same module will be strongly connected with each other.
2. The amount of intramodule connectivity in a good partition will be greater than expected by chance, as defined by a network in which edges are placed between nodes at random.
3. Proposed by Newman and Girvan, 2004.

# Configuration Model



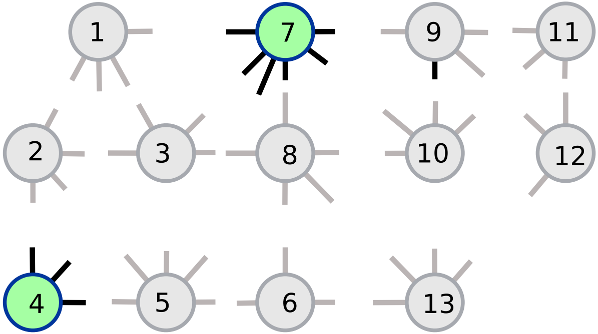

- Method to generate random graphs like Erdös-Renyi and Watts-Strogatz models.
- Ensure that the random graphs have the same degree sequence as $G$, but allow self loops and multi-edges.

# Configuration Model



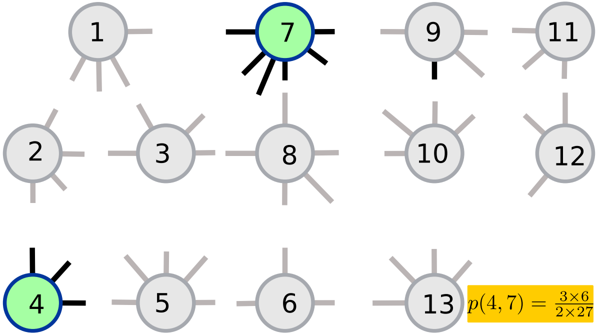

- Cut each edge in $G$ in half.
- Each node $u$ has $d(u)$ stubs; total number of stubs is $2m$.
- For each stub select another stub uniformly at random and connect them by an edge.

# Configuration Model



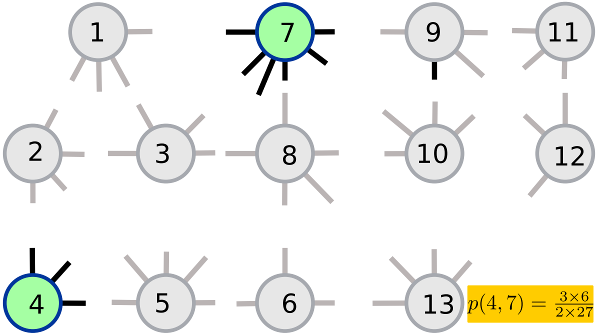

- What is the probability of an edge between nodes $u$ and $v$?

# Configuration Model




- What is the probability of an edge between nodes $u$ and $v$? $\frac{d(u)d(v)}{2m}$.

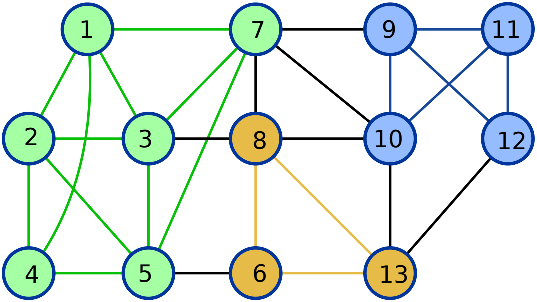
# Configuration Model




- What is the probability of an edge between nodes $u$ and $v$? $\frac{d(u)d(v)}{2m}$.
- Therefore modularity of the partition of a random graph in the configuration model into the same modules $\mathcal{C} = C_1, C_2, \ldots C_k$

$$q(\mathcal{C}) = \frac{1}{2m} \sum_{u,v \in V} \frac{d(u)d(v)}{2m} \delta(c(u), c(v))$$

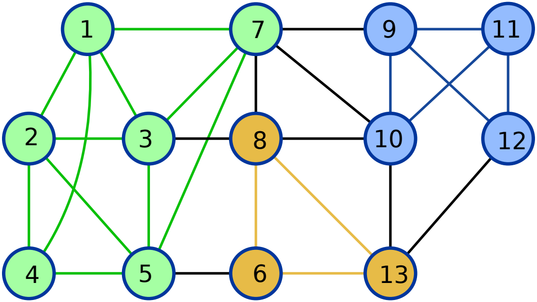
# Final Definition of Modularity




$$q(\mathcal{C}) = \frac{1}{2m} \sum_{u,v \in V} \left( a(u,v) - \frac{d(u)d(v))}{2m} \right) \delta(C(u), C(v))$$

- What is the range of $q(\mathcal{C})$?

# Final Definition of Modularity




$$q(\mathcal{C}) = \frac{1}{2m} \sum_{u,v \in V} \left( a(u,v) - \frac{d(u)d(v))}{2m} \right) \delta(C(u), C(v))$$

- What is the range of $q(\mathcal{C})$? Between -1 and 1.
  - $q(\mathcal{C}) > 0$: $\mathcal{C}$ has higher intramodule connectivity than expected by chance from configuration model.
  - $q(\mathcal{C}) = 0$: $\mathcal{C}$ has same intramodule connectivity as expected in a random graph.
  - $q(\mathcal{C}) < 0$: $\mathcal{C}$ has no modular structure.

# Using Modularity

- Now that we have defined a nice measure for the quality of a partition, how do we use it?
- Definition of $q$ does not specify the number of clusters.

# Using Modularity

- Now that we have defined a nice measure for the quality of a partition, how do we use it?
- Definition of $q$ does not specify the number of clusters.
- Hierarchical clustering: Compute modularity after every merge and output the clustering with the largest value.
- Any other clustering algorithm: compute the modularity of the result.

# Using Modularity

- Now that we have defined a nice measure for the quality of a partition, how do we use it?
- Definition of $q$ does not specify the number of clusters.
- Hierarchical clustering: Compute modularity after every merge and output the clustering with the largest value.
- Any other clustering algorithm: compute the modularity of the result.
- Develop a new algorithm to maximise modularity.
  - Maximising modularity is NP-hard.
  - We must rely on heuristics to make the modularity as large as possible.

# Greedy Algorithm

- Proposed by Newman, 2004.

1. Start with every node in its own module.
2. While there are at least two modules
    1. Compute the pair of modules whose merger will result in the largest increase or smallest decrease in $q$.
    2. Merge this pair of modules into one.
3. Return the clustering with the largest value of $q$.

# Greedy Algorithm

- Proposed by Newman, 2004.

1. Start with every node in its own module.
2. While there are at least two modules
    1. Compute the pair of modules whose merger will result in the largest increase or smallest decrease in $q$.
    2. Merge this pair of modules into one.
3. Return the clustering with the largest value of $q$.

- Hierarchical clustering algorithm built directly around maximisation of $q$.
- Allows $q$ to decrease to preserve the principle of hierarchical clustering.
- Why is the algorithm "greedy"?

# Greedy Algorithm

- Proposed by Newman, 2004.

1. Start with every node in its own module.
2. While there are at least two modules
    1. Compute the pair of modules whose merger will result in the largest increase or smallest decrease in $q$.
    2. Merge this pair of modules into one.
3. Return the clustering with the largest value of $q$.

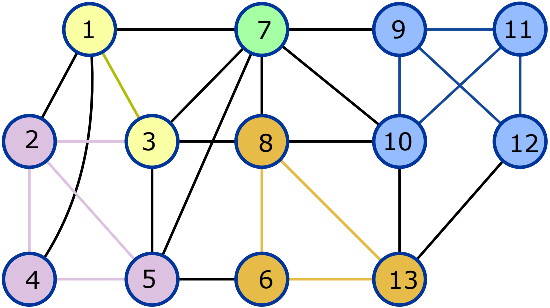
- Hierarchical clustering algorithm built directly around maximisation of $q$.
- Allows $q$ to decrease to preserve the principle of hierarchical clustering.
- Why is the algorithm "greedy"? Merging of two modules cannot be undone.

# Louvain Algorithm: Phase 1

- Proposed by Blondel *et al.*, 2008.

1. Start with every node in its own module.
2. For every node $u \in V$ and every neighbour $v$ of $u$, evaluate the change in $q$ when we remove $u$ from its module and add it to $v$'s module.
3. Move $u$ to that neighbour's module for which increase in $q$ is largest.
4. Repeat the previous two steps until $q$ does not increase.

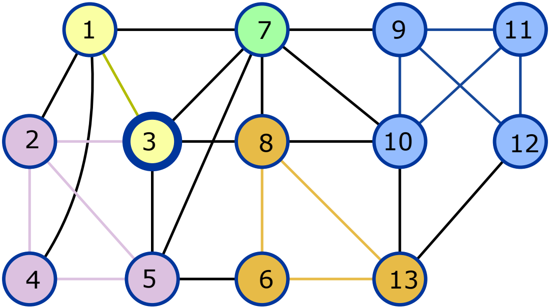
# Louvain Algorithm: Phase 1




- Proposed by Blondel *et al.*, 2008.

1. Start with every node in its own module.
2. For every node $u \in V$ and every neighbour $v$ of $u$, evaluate the change in $q$ when we remove $u$ from its module and add it to $v$'s module.
3. Move $u$ to that neighbour's module for which increase in $q$ is largest.
4. Repeat the previous two steps until $q$ does not increase.

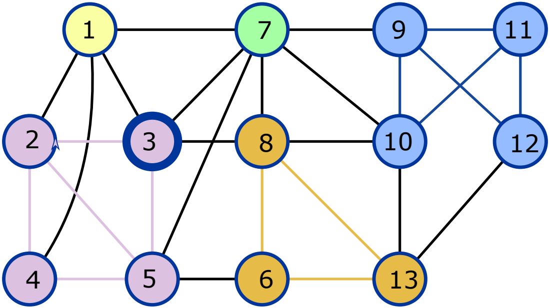
# Louvain Algorithm: Phase 1




- Proposed by Blondel *et al.*, 2008.

1. Start with every node in its own module.
2. For every node $u \in V$ and every neighbour $v$ of $u$, evaluate the change in $q$ when we remove $u$ from its module and add it to $v$'s module.
3. Move $u$ to that neighbour's module for which increase in $q$ is largest.
4. Repeat the previous two steps until $q$ does not increase.

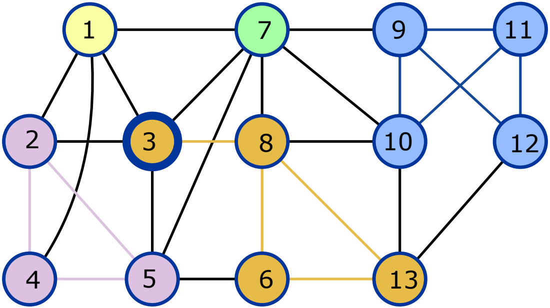
# Louvain Algorithm: Phase 1




- Proposed by Blondel *et al.*, 2008.

1. Start with every node in its own module.
2. For every node $u \in V$ and every neighbour $v$ of $u$, evaluate the change in $q$ when we remove $u$ from its module and add it to $v$'s module.
3. Move $u$ to that neighbour's module for which increase in $q$ is largest.
4. Repeat the previous two steps until $q$ does not increase.

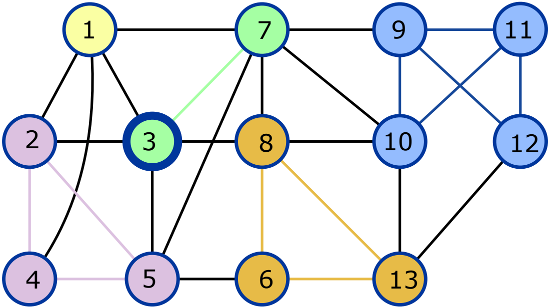
# Louvain Algorithm: Phase 1




- Proposed by Blondel *et al.*, 2008.

1. Start with every node in its own module.
2. For every node $u \in V$ and every neighbour $v$ of $u$, evaluate the change in $q$ when we remove $u$ from its module and add it to $v$'s module.
3. Move $u$ to that neighbour's module for which increase in $q$ is largest.
4. Repeat the previous two steps until $q$ does not increase.

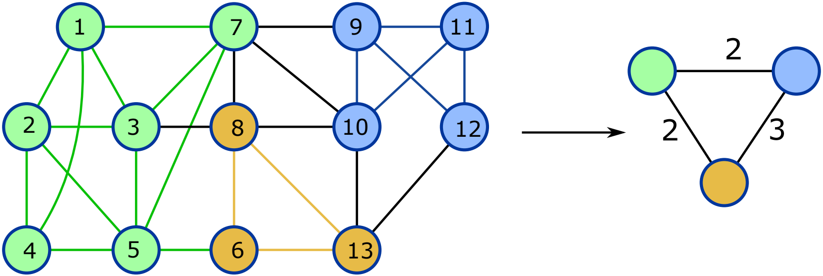
# Louvain Algorithm: Phase 1




- Proposed by Blondel *et al.*, 2008.

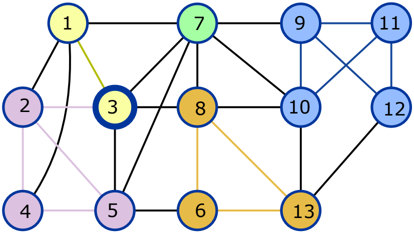
1. Start with every node in its own module.
2. For every node $u \in V$ and every neighbour $v$ of $u$, evaluate the change in $q$ when we remove $u$ from its module and add it to $v$'s module.
3. Move $u$ to that neighbour's module for which increase in $q$ is largest.
4. Repeat the previous two steps until $q$ does not increase.

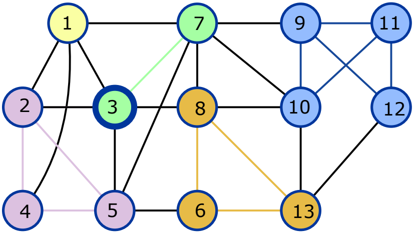
# Louvain Algorithm: Phase 2



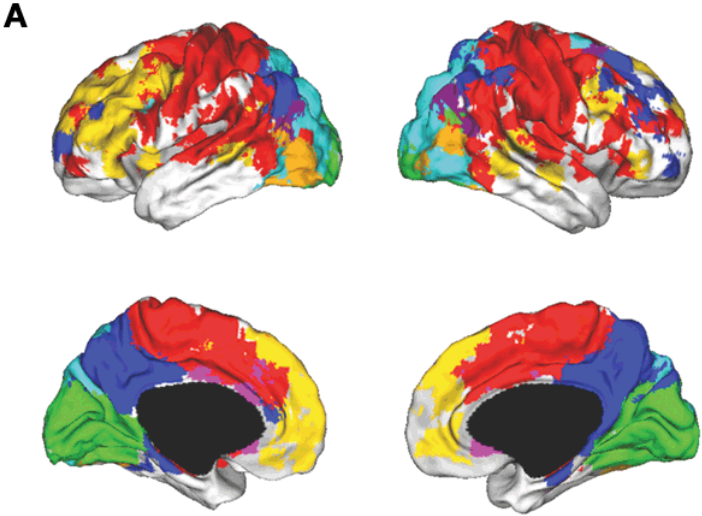

1. Construct a new graph where every module is a node and a weighted edge represents (multiple) connections between two modules.
2. Repeat Phases 1 and 2 until no further gains in $q$ are possible.

# Louvain Algorithm: Efficiency



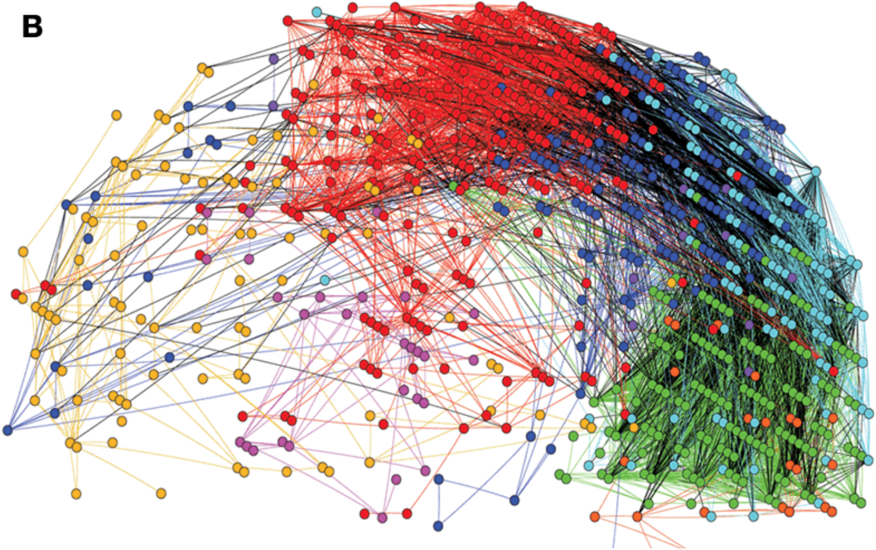




- Efficient calculation of change in $q$ upon swapping makes this algorithm very fast.



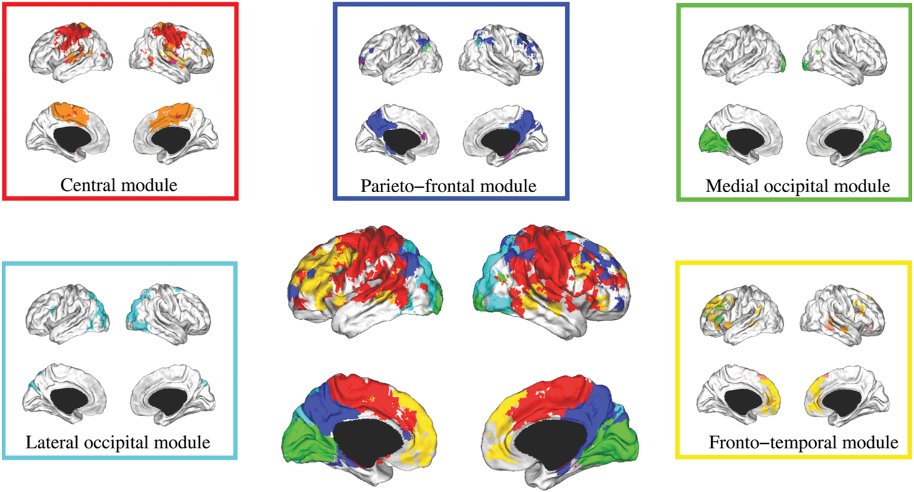

Human resting-state fMRI networks, 1,800 nodes, $4mm^3$ voxels, had three hierarchical levels: eight modules at the highest level, each with $> 10$ nodes, 57 modules at the lowest level of the hierarchy.

Meunier *et al.*, 2009



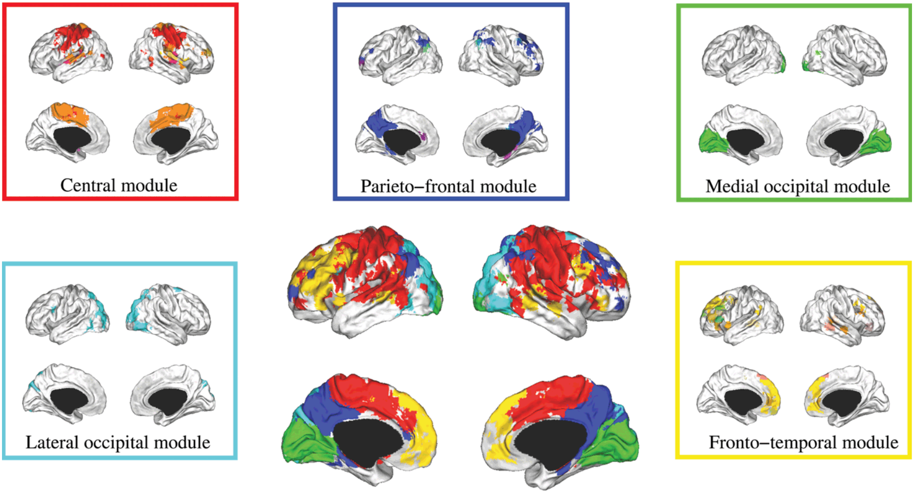

Visualisation of modules. View of brain is from the left side with the frontal cortex on the left and the occipital cortex on the right.

Meunier *et al.*, 2009



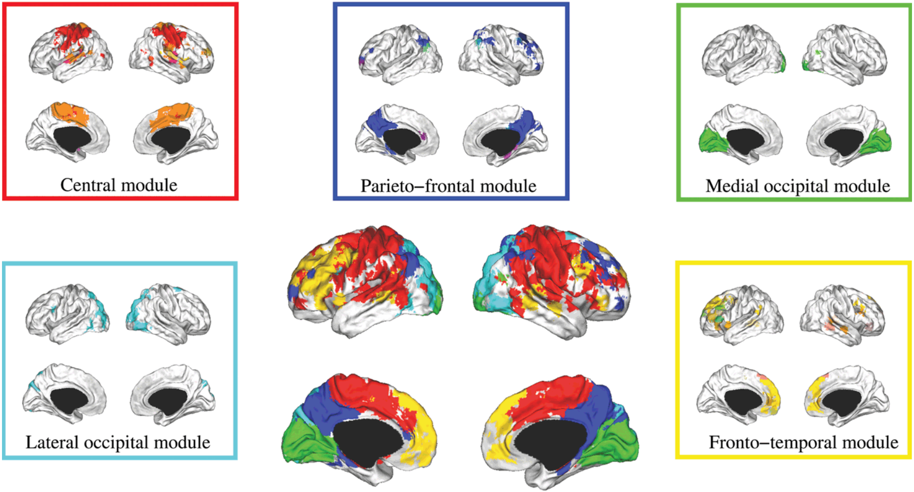

Decomposition of the five largest modules (in the centre): medial occipital module has no major sub-modules whereas the fronto-temporal module has many sub-modules.

Meunier *et al.*, 2009




Medial occipital module (primary visual): This module comprised medial occipital cortex and occipital pole, including primary visual areas.

Meunier *et al.*, 2009




Fronto-temporal module (symbolic): less symmetrically organized than most of the other high level modules and contained larger number of sub-modules at lower levels.

Meunier *et al.*, 2009

# Limitations of Modularity

- Modularity generally increases as number of nodes and modules in a graph increase.
- Many very similar partitions have similar values of $q$.
- Modularity has a resolution limit: small modules may be combined simply to increase $q$. (Read Box 9.2 in the textbook.)
- Random graph model is quite simple: assumes every node has an equal probability of connecting to every other node.

# Limitations of Modularity

- Modularity generally increases as number of nodes and modules in a graph increase.
- Many very similar partitions have similar values of $q$.
- Modularity has a resolution limit: small modules may be combined simply to increase $q$. (Read Box 9.2 in the textbook.)
- Random graph model is quite simple: assumes every node has an equal probability of connecting to every other node.
- Many alternatives proposed to address these limitations.