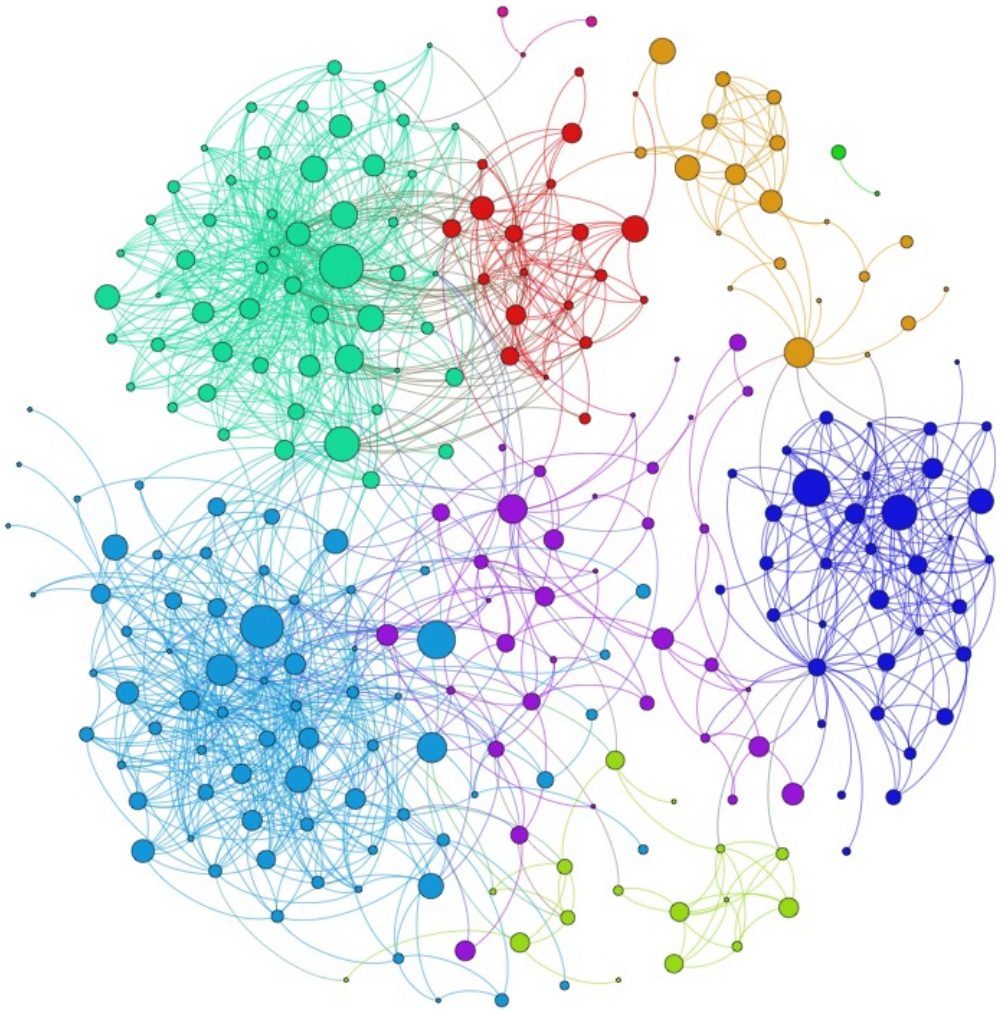
Graph Mining and Community Detection

Pedro L. C. Rodrigues & Jean-Baptiste Durand

Credits: Charlotte Laclau

Community detection $\equiv$ Clustering of graph vertices

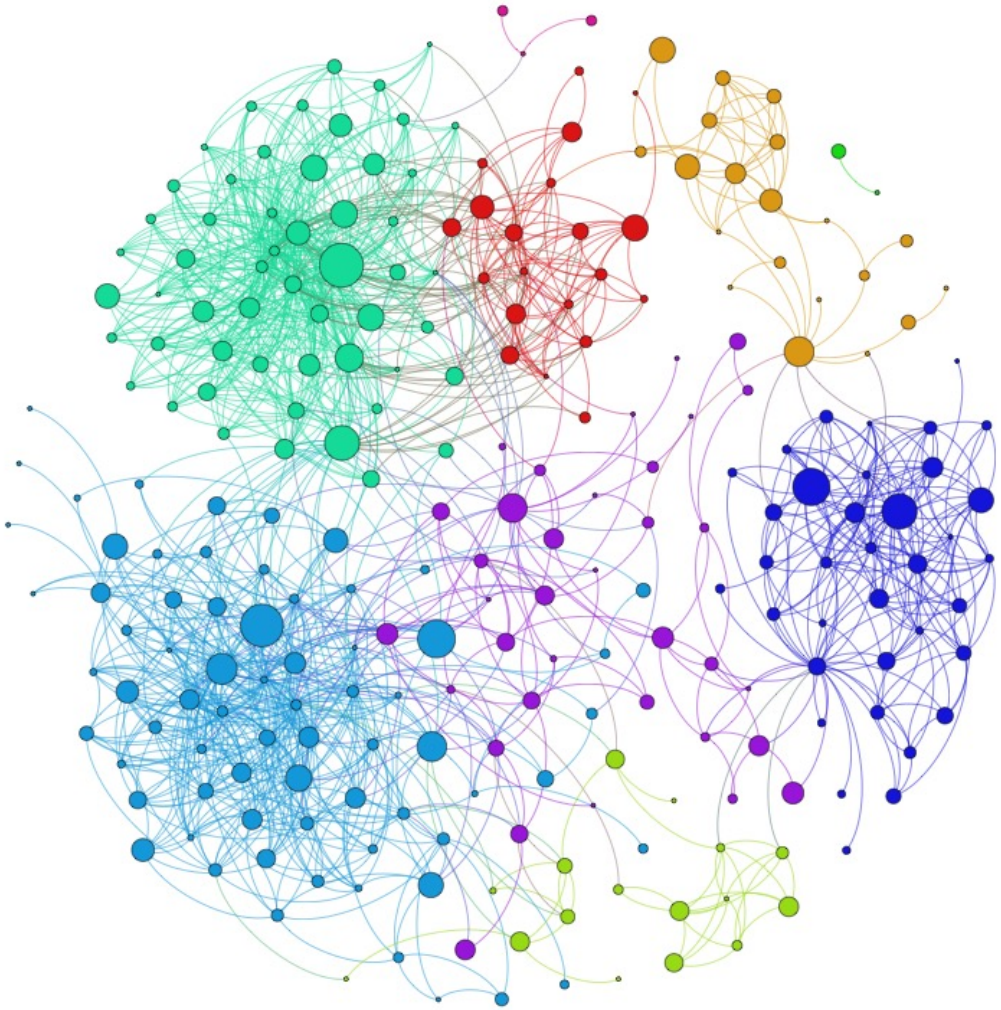
Example 1: Social network graph

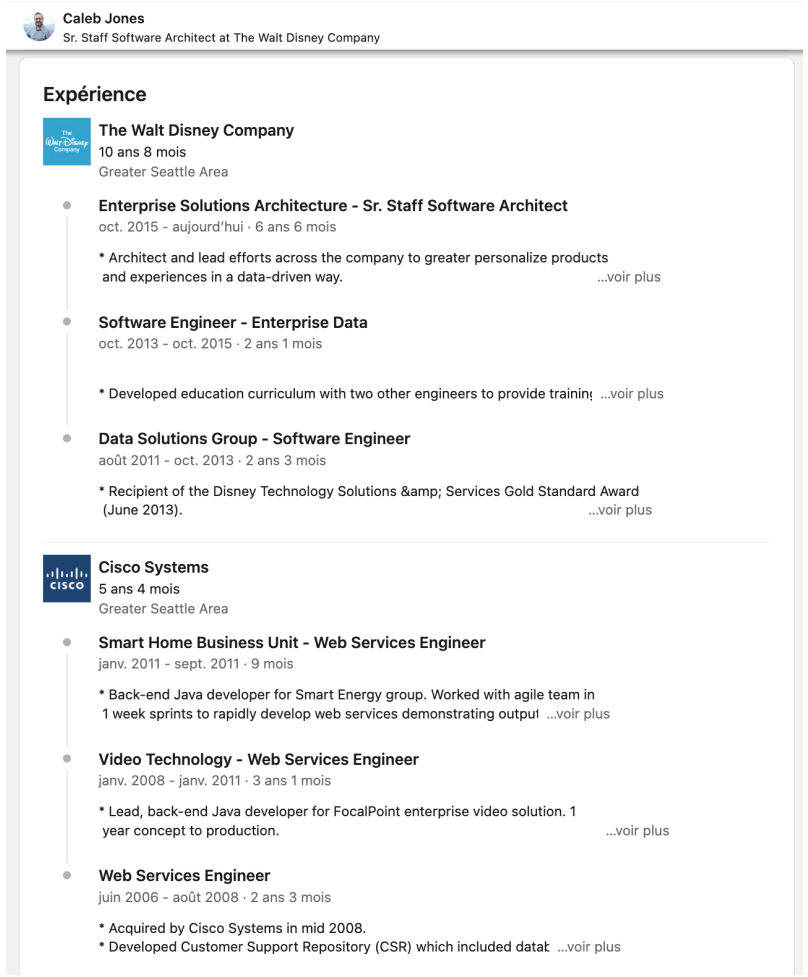


LinkedIn graph of connections of Caleb Jones in 2014

Reference: **allthingsgraphed.com/2014/10/16/your-linkedin-network/**

Example 1: Social network graph





Caleb Jones
Sr. Staff Software Architect at The Walt Disney Company

Expérience

The Walt Disney Company
10 ans 8 mois
Greater Seattle Area

Enterprise Solutions Architecture - Sr. Staff Software Architect
oct. 2015 - aujourd'hui · 6 ans 6 mois

* Architect and lead efforts across the company to greater personalize products and experiences in a data-driven way. ...voir plus

Software Engineer - Enterprise Data
oct. 2013 - oct. 2015 · 2 ans 1 mois

* Developed education curriculum with two other engineers to provide trainin ...voir plus

Data Solutions Group - Software Engineer
août 2011 - oct. 2013 · 2 ans 3 mois

* Recipient of the Disney Technology Solutions & Services Gold Standard Award (June 2013). ...voir plus

Cisco Systems
5 ans 4 mois
Greater Seattle Area

Smart Home Business Unit - Web Services Engineer
janv. 2011 - sept. 2011 · 9 mois

* Back-end Java developer for Smart Energy group. Worked with agile team in 1 week sprints to rapidly develop web services demonstrating outpu ...voir plus

Video Technology - Web Services Engineer
janv. 2008 - janv. 2011 · 3 ans 1 mois

* Lead, back-end Java developer for FocalPoint enterprise video solution. 1 year concept to production. ...voir plus

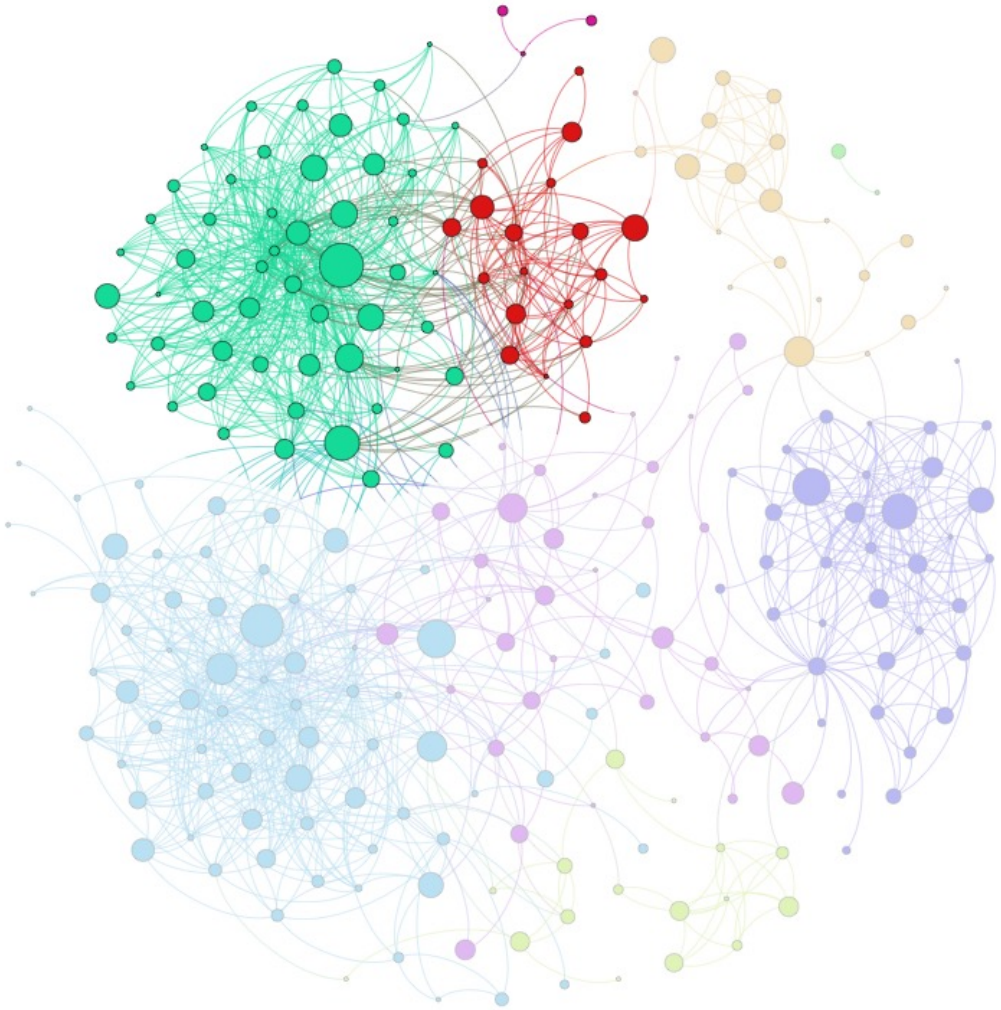
Web Services Engineer
juin 2006 - août 2008 · 2 ans 3 mois

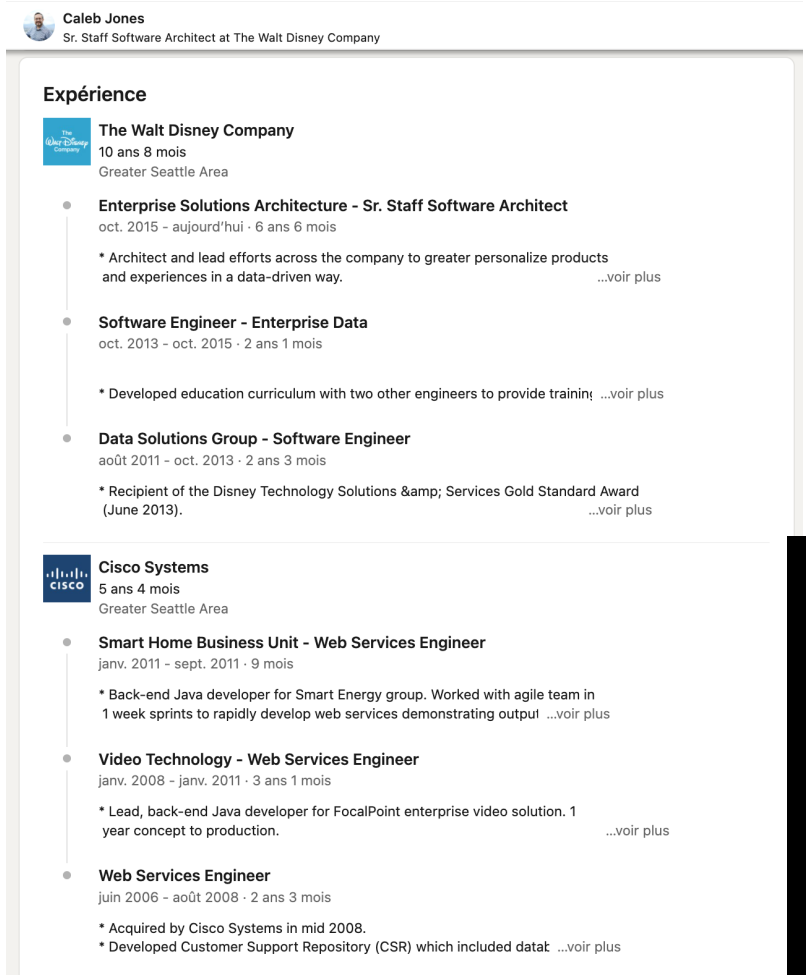
* Acquired by Cisco Systems in mid 2008.
* Developed Customer Support Repository (CSR) which included data ...voir plus

LinkedIn graph of connections of Caleb Jones in 2014

Reference: **allthingsgraphed.com/2014/10/16/your-linkedin-network/**

Example 1: Social network graph





Caleb Jones
Sr. Staff Software Architect at The Walt Disney Company

Expérience

The Walt Disney Company
10 ans 8 mois
Greater Seattle Area

Enterprise Solutions Architecture - Sr. Staff Software Architect
oct. 2015 - aujourd'hui · 6 ans 6 mois

* Architect and lead efforts across the company to greater personalize products and experiences in a data-driven way. ...voir plus

Software Engineer - Enterprise Data
oct. 2013 - oct. 2015 · 2 ans 1 mois

* Developed education curriculum with two other engineers to provide trainin ...voir plus

Data Solutions Group - Software Engineer
août 2011 - oct. 2013 · 2 ans 3 mois

* Recipient of the Disney Technology Solutions & Services Gold Standard Award (June 2013). ...voir plus

Cisco Systems
5 ans 4 mois
Greater Seattle Area

Smart Home Business Unit - Web Services Engineer
janv. 2011 - sept. 2011 · 9 mois

* Back-end Java developer for Smart Energy group. Worked with agile team in 1 week sprints to rapidly develop web services demonstrating outpu ...voir plus

Video Technology - Web Services Engineer
janv. 2008 - janv. 2011 · 3 ans 1 mois

* Lead, back-end Java developer for FocalPoint enterprise video solution. 1 year concept to production. ...voir plus

Web Services Engineer
juin 2006 - août 2008 · 2 ans 3 mois

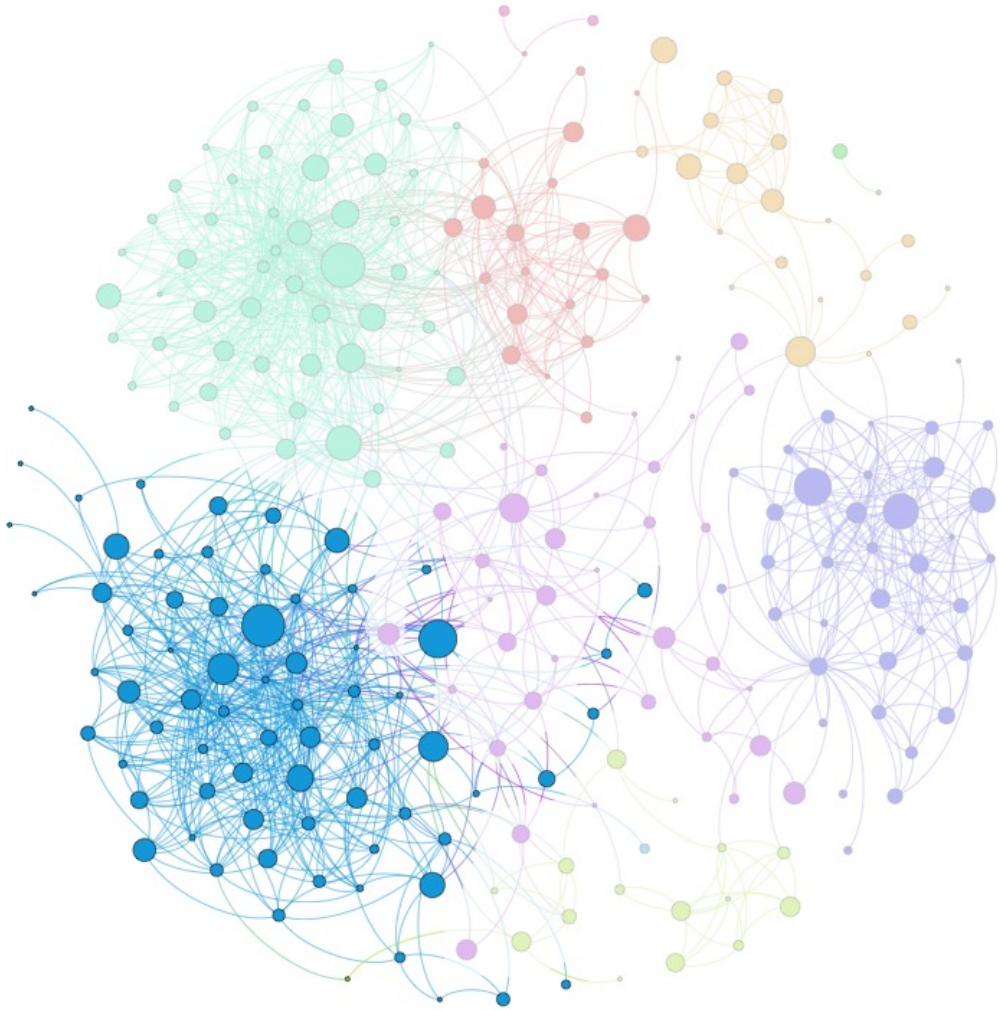
* Acquired by Cisco Systems in mid 2008.
* Developed Customer Support Repository (CSR) which included data ...voir plus

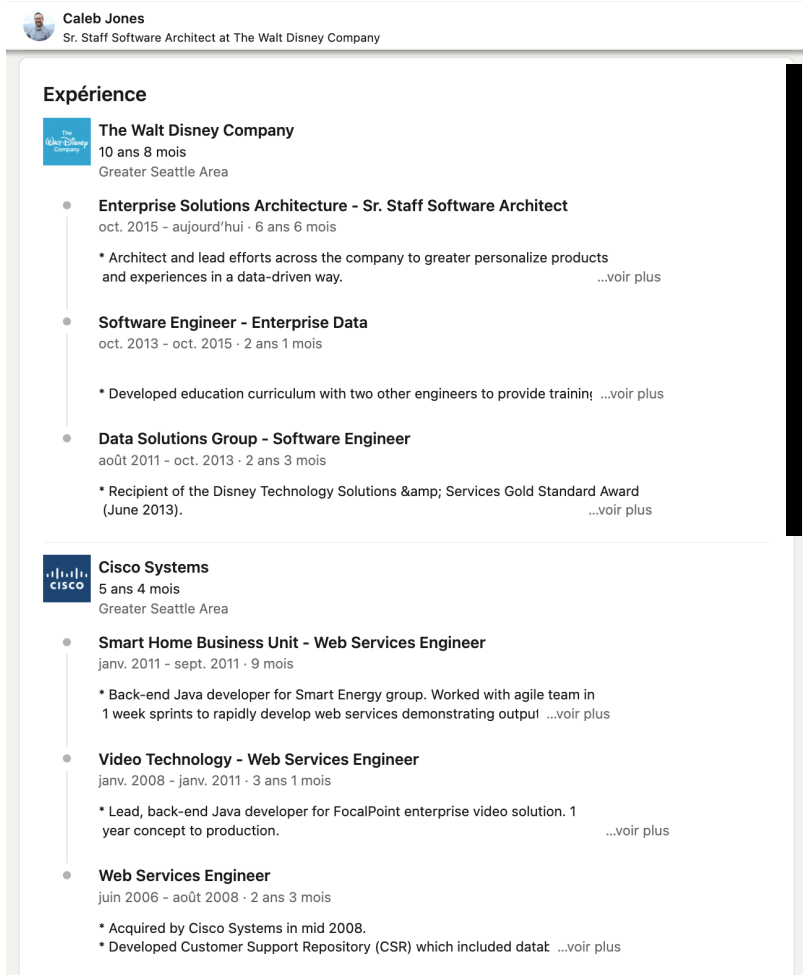
The startup where Caleb used to work (red) was acquired by Cisco (green)

LinkedIn graph of connections of Caleb Jones in 2014

Reference: allthingsgraphed.com/2014/10/16/your-linkedin-network/

Example 1: Social network graph





Caleb Jones
Sr. Staff Software Architect at The Walt Disney Company

Expérience

The Walt Disney Company
10 ans 8 mois
Greater Seattle Area

Enterprise Solutions Architecture - Sr. Staff Software Architect
oct. 2015 - aujourd'hui · 6 ans 6 mois

* Architect and lead efforts across the company to greater personalize products and experiences in a data-driven way. ...voir plus

Software Engineer - Enterprise Data
oct. 2013 - oct. 2015 · 2 ans 1 mois

* Developed education curriculum with two other engineers to provide trainin ...voir plus

Data Solutions Group - Software Engineer
août 2011 - oct. 2013 · 2 ans 3 mois

* Recipient of the Disney Technology Solutions & Services Gold Standard Award (June 2013). ...voir plus

Cisco Systems
5 ans 4 mois
Greater Seattle Area

Smart Home Business Unit - Web Services Engineer
janv. 2011 - sept. 2011 · 9 mois

* Back-end Java developer for Smart Energy group. Worked with agile team in 1 week sprints to rapidly develop web services demonstrating outpu ...voir plus

Video Technology - Web Services Engineer
janv. 2008 - janv. 2011 · 3 ans 1 mois

* Lead, back-end Java developer for FocalPoint enterprise video solution. 1 year concept to production. ...voir plus

Web Services Engineer
juin 2006 - août 2008 · 2 ans 3 mois

* Acquired by Cisco Systems in mid 2008.
* Developed Customer Support Repository (CSR) which included data ...voir plus

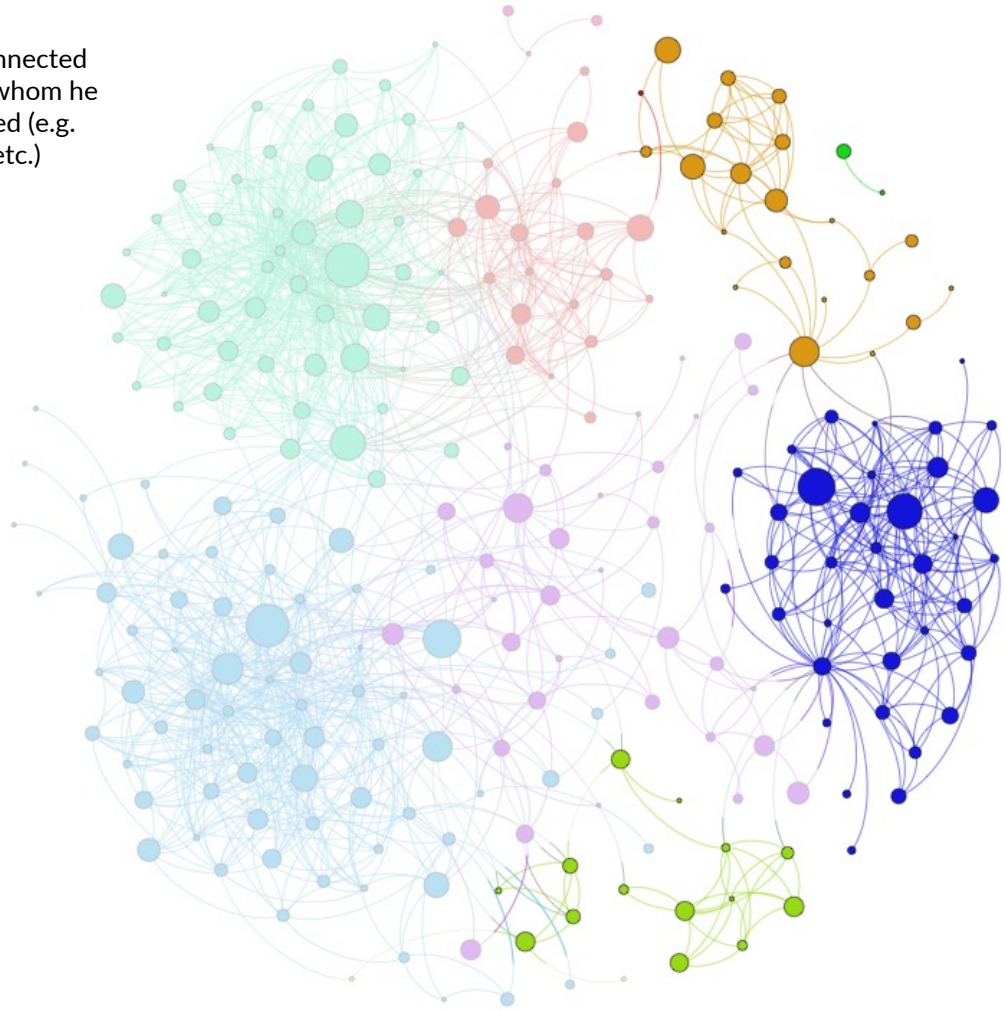
Caleb changed of job (blue) with completely new colleagues

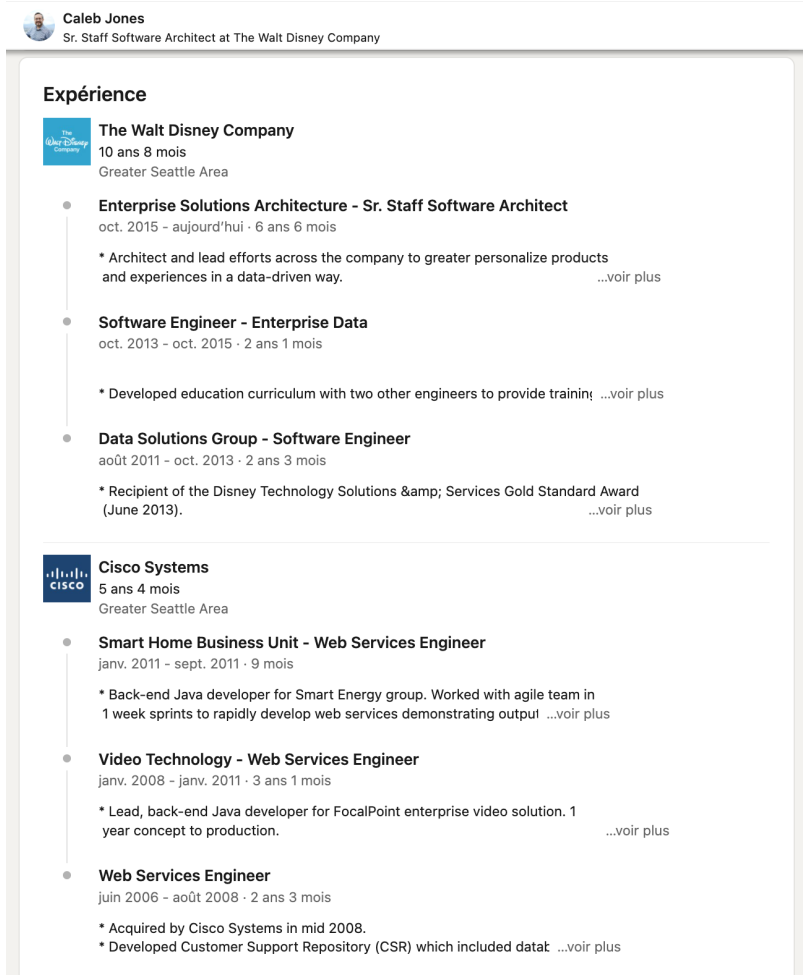
LinkedIn graph of connections of Caleb Jones in 2014

Reference: allthingsgraphed.com/2014/10/16/your-linkedin-network/

Example 1: Social network graph

Caleb is also connected to people with whom he has never worked (e.g. friends, family, etc.)





Caleb Jones
Sr. Staff Software Architect at The Walt Disney Company

Expérience

The Walt Disney Company
10 ans 8 mois
Greater Seattle Area

Enterprise Solutions Architecture - Sr. Staff Software Architect
oct. 2015 - aujourd'hui · 6 ans 6 mois

* Architect and lead efforts across the company to greater personalize products and experiences in a data-driven way. ...voir plus

Software Engineer - Enterprise Data
oct. 2013 - oct. 2015 · 2 ans 1 mois

* Developed education curriculum with two other engineers to provide trainin ...voir plus

Data Solutions Group - Software Engineer
août 2011 - oct. 2013 · 2 ans 3 mois

* Recipient of the Disney Technology Solutions & Services Gold Standard Award (June 2013). ...voir plus

Cisco Systems
5 ans 4 mois
Greater Seattle Area

Smart Home Business Unit - Web Services Engineer
janv. 2011 - sept. 2011 · 9 mois

* Back-end Java developer for Smart Energy group. Worked with agile team in 1 week sprints to rapidly develop web services demonstrating outpu ...voir plus

Video Technology - Web Services Engineer
janv. 2008 - janv. 2011 · 3 ans 1 mois

* Lead, back-end Java developer for FocalPoint enterprise video solution. 1 year concept to production. ...voir plus

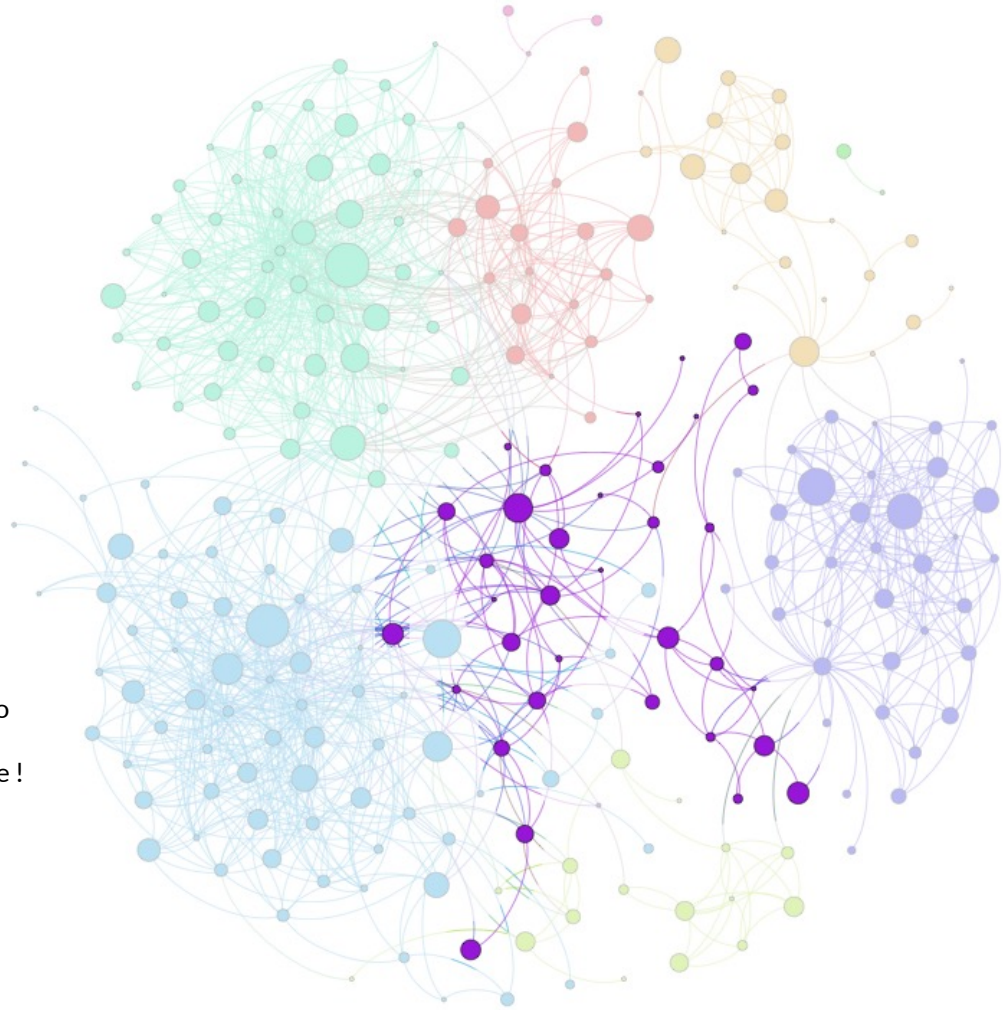
Web Services Engineer
juin 2006 - août 2008 · 2 ans 3 mois

* Acquired by Cisco Systems in mid 2008.
* Developed Customer Support Repository (CSR) which included data ...voir plus

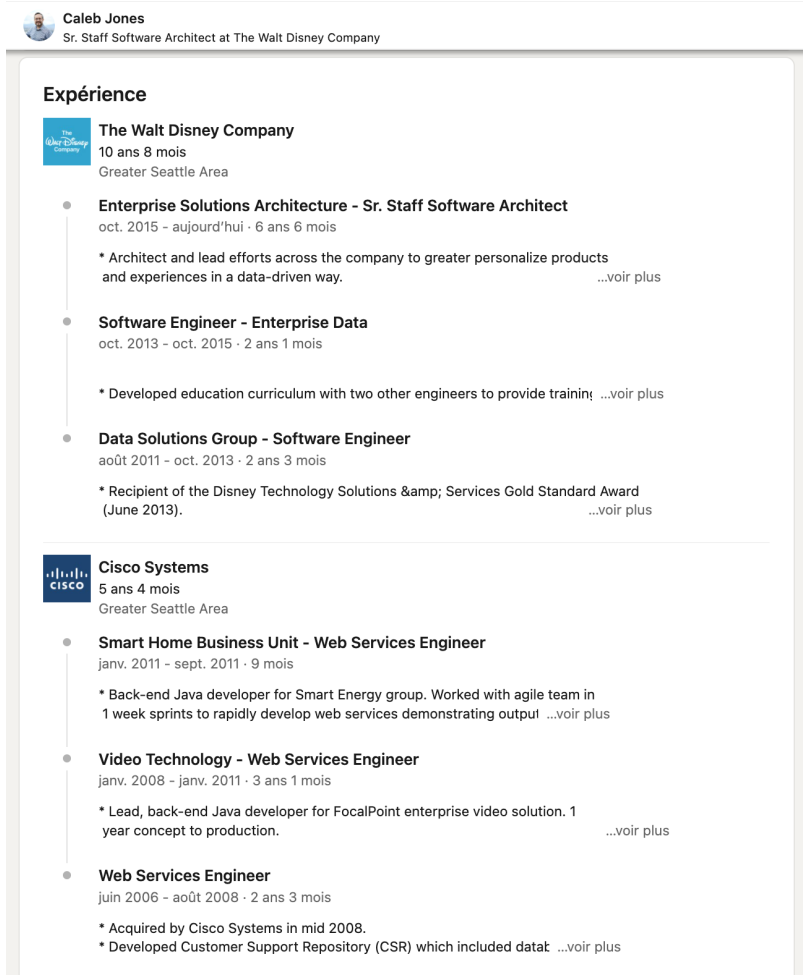
LinkedIn graph of connections of Caleb Jones in 2014

Reference: **allthingsgraphed.com/2014/10/16/your-linkedin-network/**

Example 1: Social network graph



Recruiters (**purple**) try to have connections with people from everywhere !



Caleb Jones
Sr. Staff Software Architect at The Walt Disney Company

Expérience

The Walt Disney Company
10 ans 8 mois
Greater Seattle Area

Enterprise Solutions Architecture - Sr. Staff Software Architect
oct. 2015 - aujourd'hui · 6 ans 6 mois

* Architect and lead efforts across the company to greater personalize products and experiences in a data-driven way. ...voir plus

Software Engineer - Enterprise Data
oct. 2013 - oct. 2015 · 2 ans 1 mois

* Developed education curriculum with two other engineers to provide trainin ...voir plus

Data Solutions Group - Software Engineer
août 2011 - oct. 2013 · 2 ans 3 mois

* Recipient of the Disney Technology Solutions & Services Gold Standard Award (June 2013). ...voir plus

Cisco Systems
5 ans 4 mois
Greater Seattle Area

Smart Home Business Unit - Web Services Engineer
janv. 2011 - sept. 2011 · 9 mois

* Back-end Java developer for Smart Energy group. Worked with agile team in 1 week sprints to rapidly develop web services demonstrating outpu ...voir plus

Video Technology - Web Services Engineer
janv. 2008 - janv. 2011 · 3 ans 1 mois

* Lead, back-end Java developer for FocalPoint enterprise video solution. 1 year concept to production. ...voir plus

Web Services Engineer
juin 2006 - août 2008 · 2 ans 3 mois

* Acquired by Cisco Systems in mid 2008.
* Developed Customer Support Repository (CSR) which included data ...voir plus

LinkedIn graph of connections of Caleb Jones in 2014

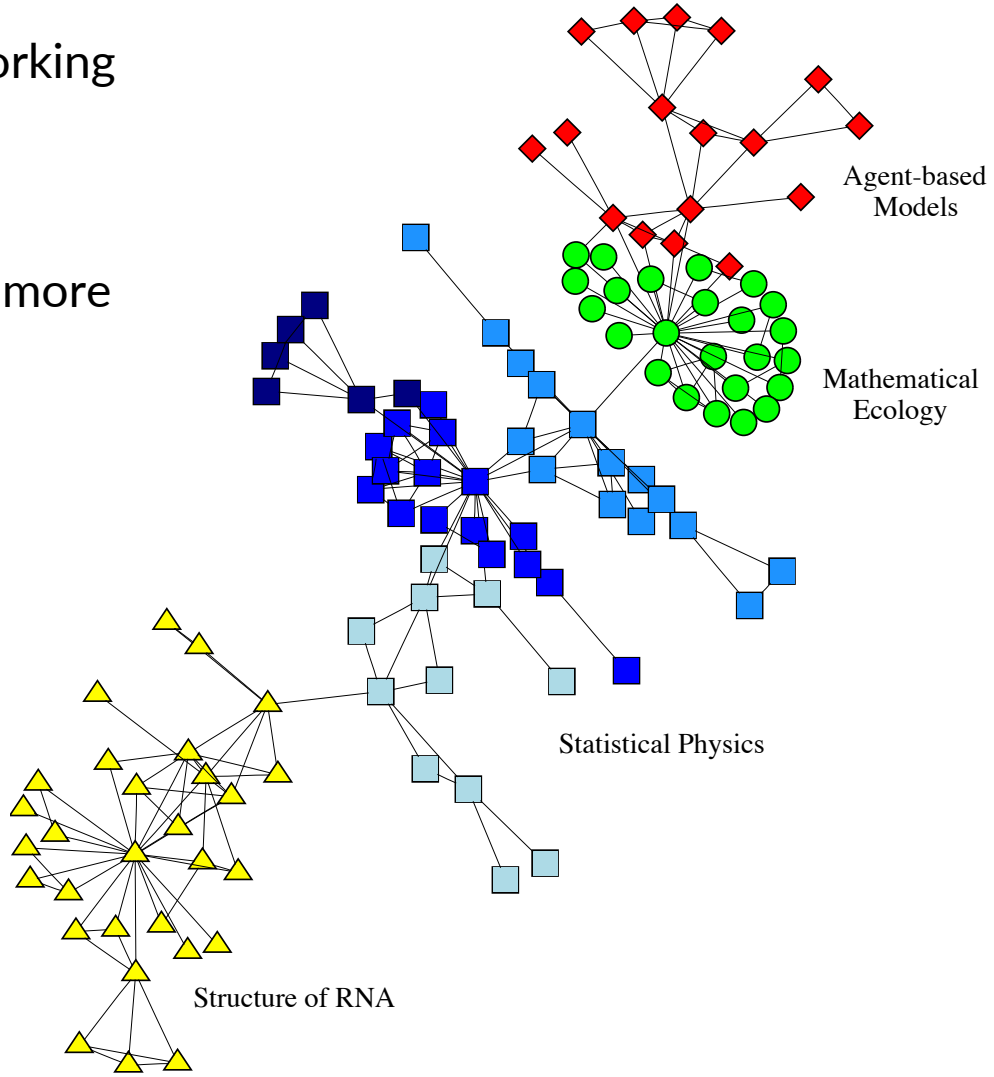
Reference: **allthingsgraphed.com/2014/10/16/your-linkedin-network/**

Example 2: Graph of scientific collaborations

Each vertex represents a scientist working at the Santa Fé Institute in the US.

An edge is drawn between a pair of scientists if they co-authored one or more articles between 2009 and 2000.

The colors correspond to different scientific domains.



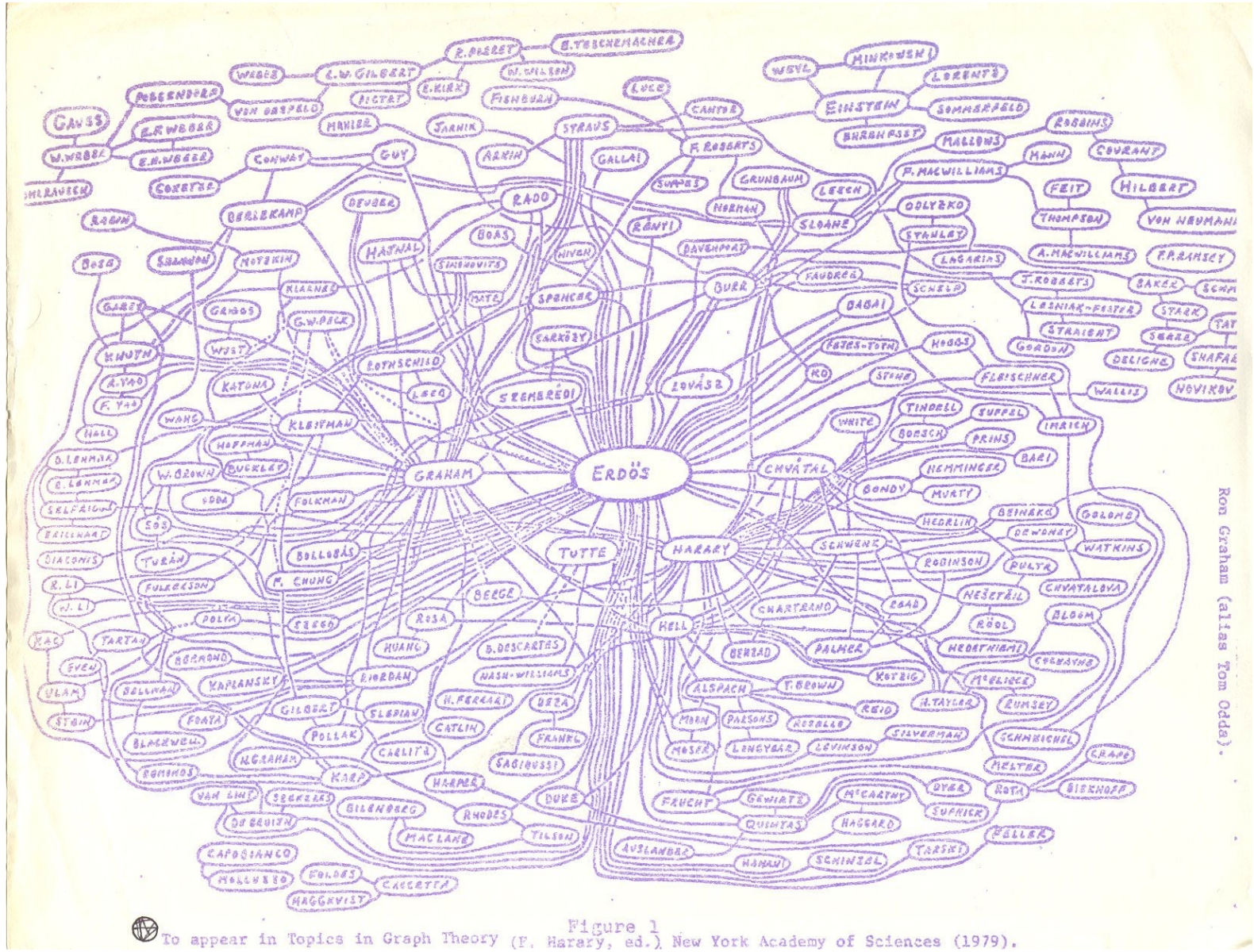
Co-authors interaction graph

Reference: Girvan “Community structure in social and biological networks”

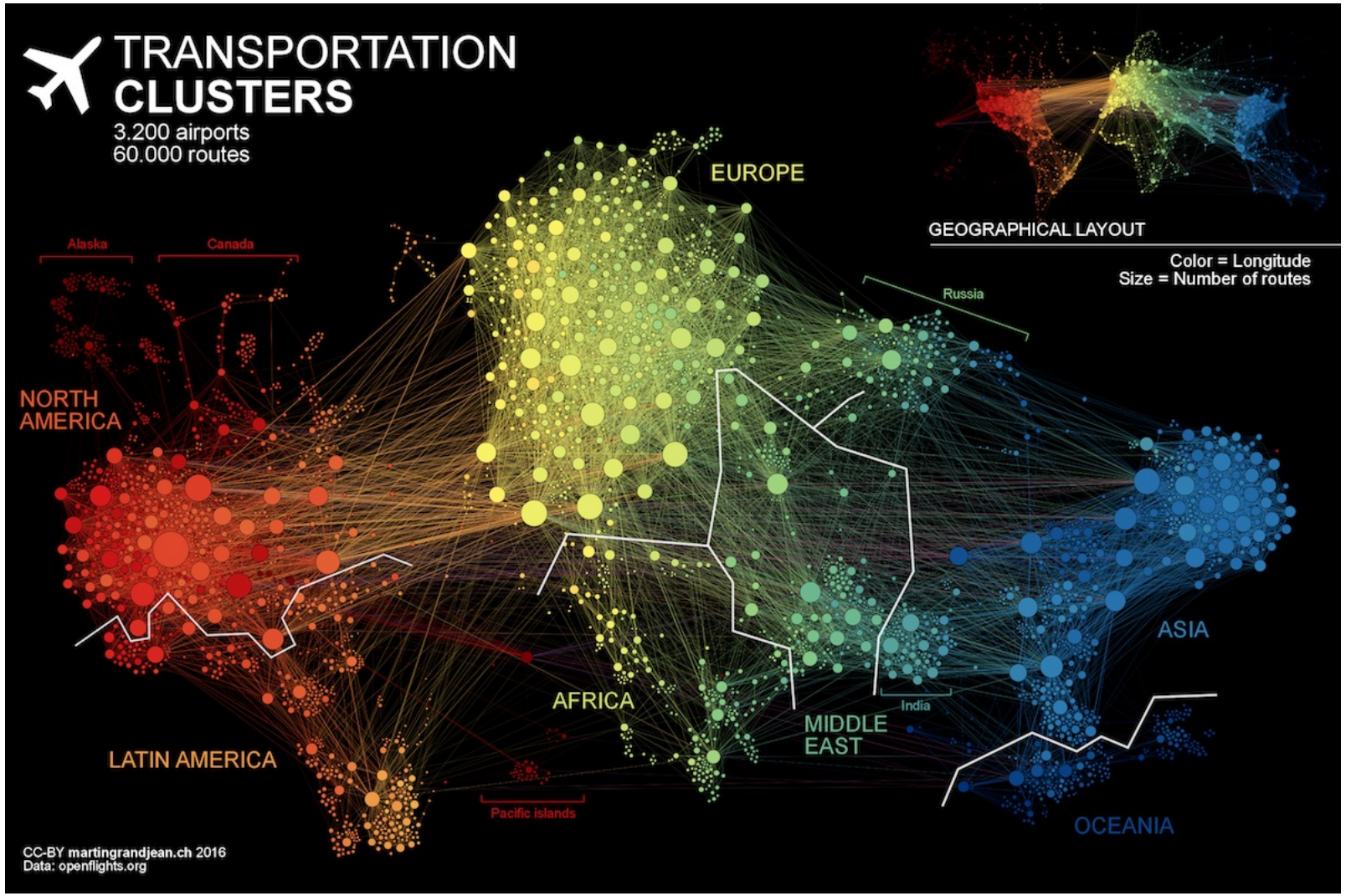
Example 2: Graph of scientific collaborations

“The Erdős number is the number of hops needed to connect the author of a paper with the prolific late mathematician Paul Erdős. An author's Erdős number is 1 if he has co-authored a paper with Erdős, 2 if he has co-authored a paper with someone who has co-authored a paper with Erdős, etc.”

mathworld.wolfram.com



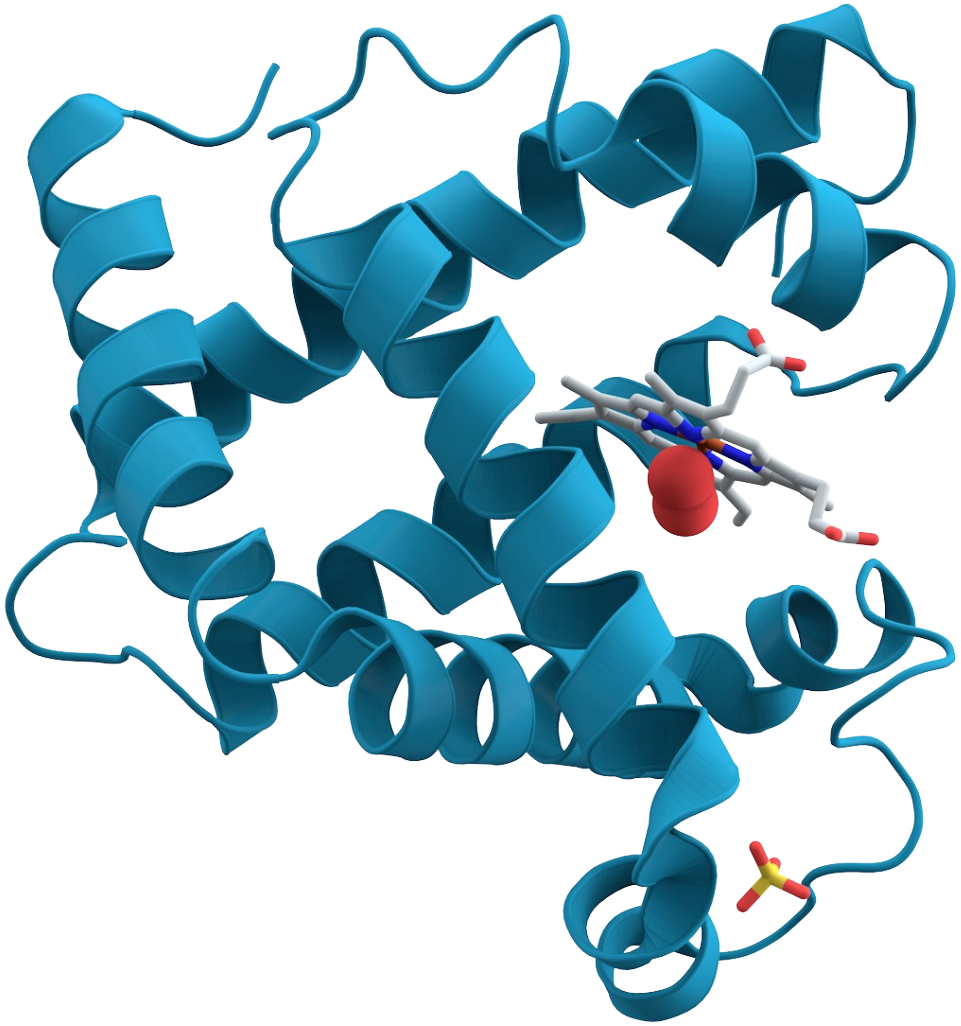
Example 3: Transportation network



Graph of airplane routes between airports in the world

Reference: www.martingrandjean.ch/connected-world-air-traffic-network/

Example 4: Chemistry



The molecular structure of protein myogloblin

Reference: **en.wikipedia.org/wiki/Myoglobin**

AlphaFold: a solution to a 50-year-old grand challenge in biology

Reference: **deepmind.com/blog/article/alphafold-a-solution-to-a-50-year-old-grand-challenge-in-biology**

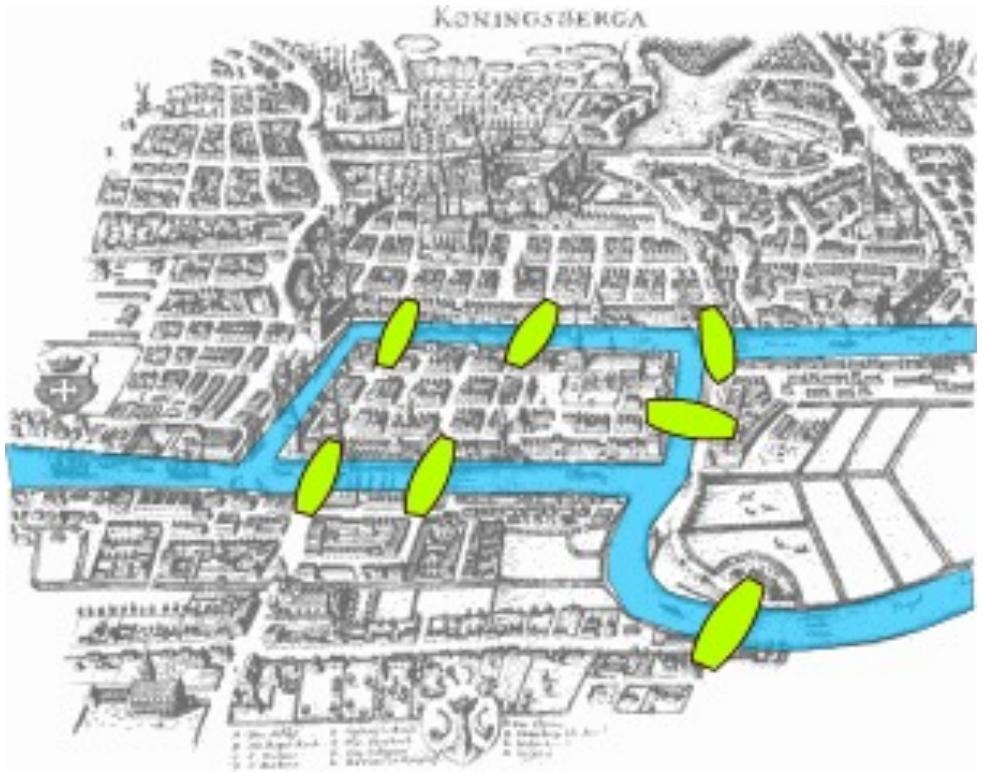
Part 1: Graphs basics

Part 2: Community detection

Part 1: Graphs basics

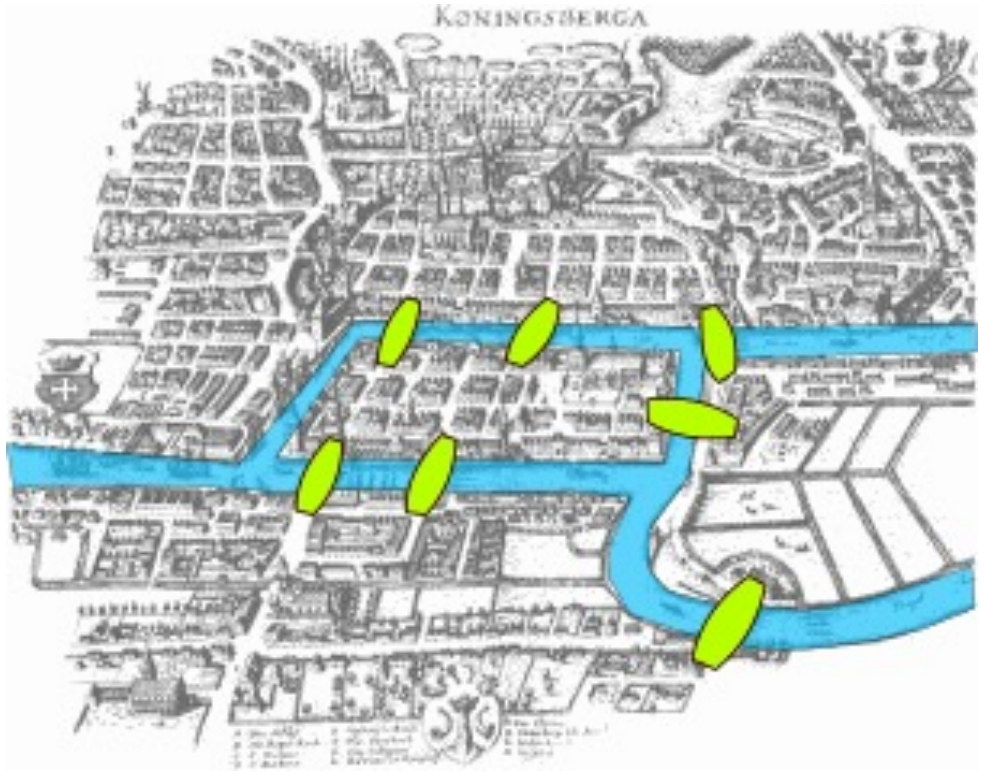
Part 2: Community detection

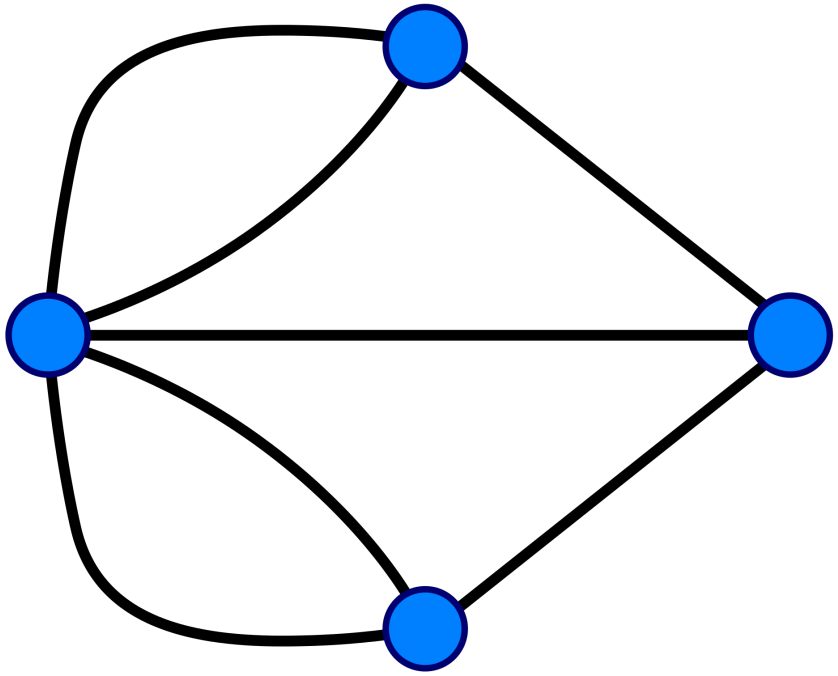
Context: The town of Königsberg is built around two islands connected by a bridge, and there exists 6 other bridges to connect these two islands.



Question: Can we walk through the city crossing all the bridges once and only once?

Context: The town of Königsberg is built around two islands connected by a bridge, and there exists 6 other bridges to connect these two islands.



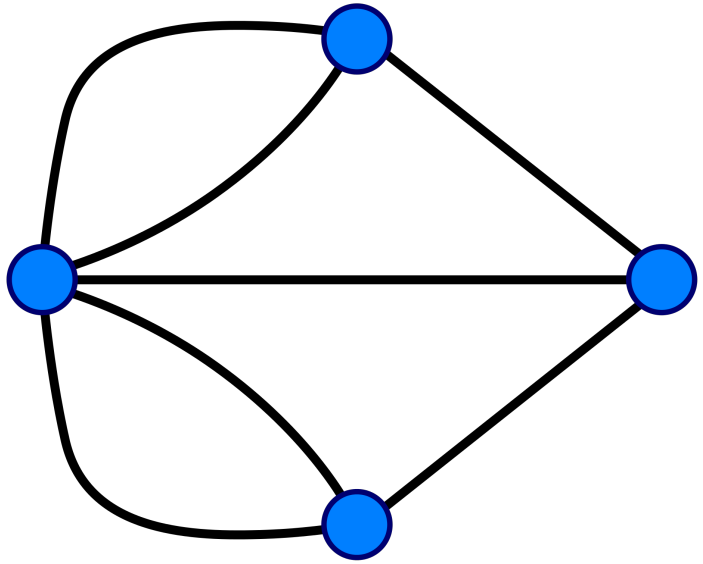


Question: Can we walk through the city crossing all the bridges once and only once?

A graph is denoted by **G = (V, E)** where

- **V** is the finite set of vertices (nodes)
- **E** is the finite set of edges (arcs)

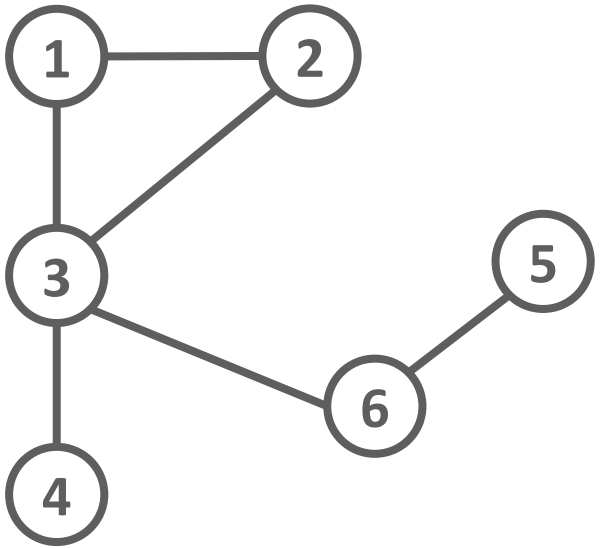
Example: {**a**, **b**} is the edge connecting vertices **a** and **b**



- How many nodes?
- How many edges?

Adjacency matrix

A graph can be represented by an **adjacency matrix** that indicates the connections between the nodes. This matrix is a 2D array of size ***n***-by-***n*** where n is the number of nodes of the graph.



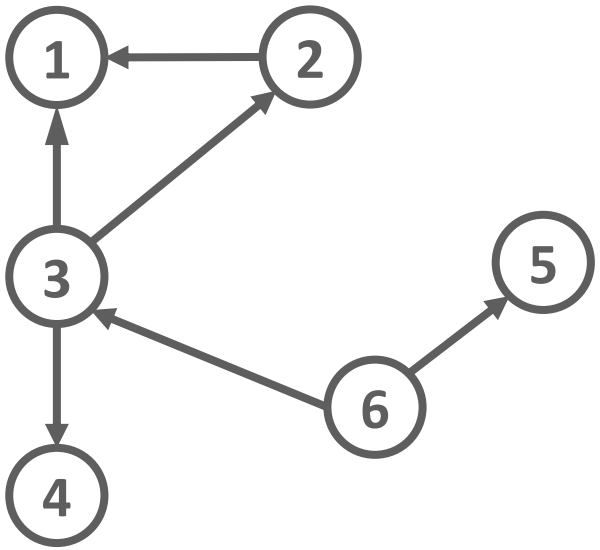
$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

$$A_{ij} = \begin{cases} 1 & \text{if } v_i \text{ and } v_j \text{ are connected} \\ 0 & \text{otherwise} \end{cases}$$

Notice that the adjancency matrix is **symmetric**

Adjacency matrix

A graph can be represented by an **adjacency matrix** that indicates the connections between the nodes. This matrix is a 2D array of size ***n***-by-***n*** where n is the number of nodes of the graph.

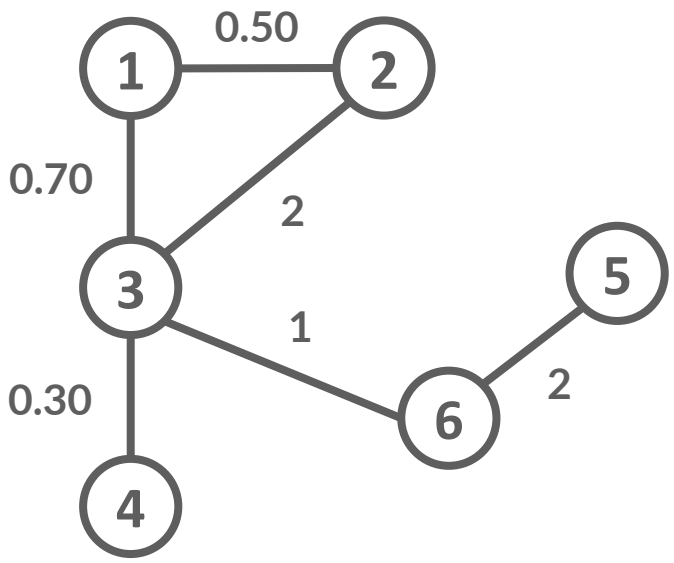


$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

In a **directed** graph the adjacency matrix is not necessarily always symmetric

Adjacency matrix

A graph can be represented by an **adjacency matrix** that indicates the connections between the nodes. This matrix is a 2D array of size ***n***-by-***n*** where n is the number of nodes of the graph.

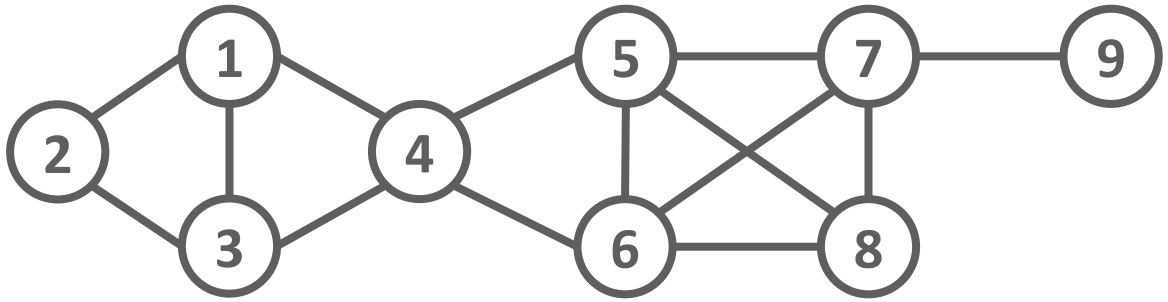


$$A = \begin{bmatrix} 0 & 0.5 & 0.7 & 0 & 0 & 0 \\ 0.5 & 0 & 2 & 0 & 0 & 0 \\ 0.7 & 2 & 0 & 0.3 & 0 & 1 \\ 0 & 0 & 0.3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & 2 & 0 \end{bmatrix}$$

In a **weighted** graph the adjacency matrix is a function of the vertices $A_{ij} = f(i, j)$

Key notions: definition and properties (1)

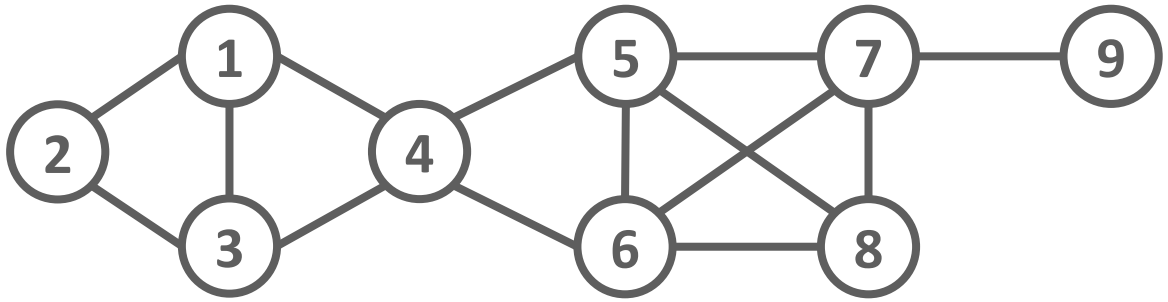
Consider the following graph **G = (V, E)** where **|V| = 9** and **|E| = 14**



Path: sucession of vertices and edges that allow to go from one vertex to another. A path **P = (V (P), E(P))** is a subgraph such that

$$V(P) = \{v^{i0}, \ldots, v^{ik}\} \quad \text{and} \quad E(P) = \{(v^{i_0}, v^{i_1}), \ldots, (v^{i_{k-1}}, v^{i_k})\}$$

Consider the following graph **G = (V, E)** where **|V| = 9** and **|E| = 14**



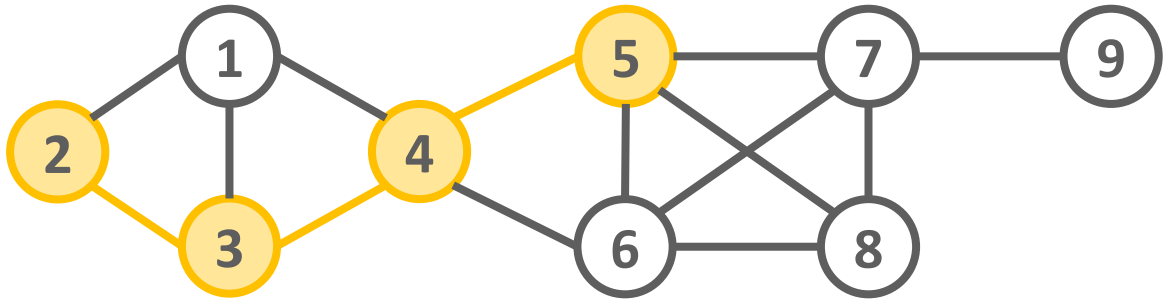
Path: sucession of vertices and edges that allow to go from one vertex to another. A path **P = (V (P), E(P))** is a subgraph such that

$$V(P) = \{v^{i0}, \ldots, v^{ik}\} \quad \text{and} \quad E(P) = \{(v^{i_0}, v^{i_1}), \ldots, (v^{i_{k-1}}, v^{i_k})\}$$

Geodesic distance: number of edges of the shortest path between two vertices

Example: in the graph above, what is the geodesic between vertices 2 and 5 ?

Consider the following graph **G = (V, E)** where **|V| = 9** and **|E| = 14**



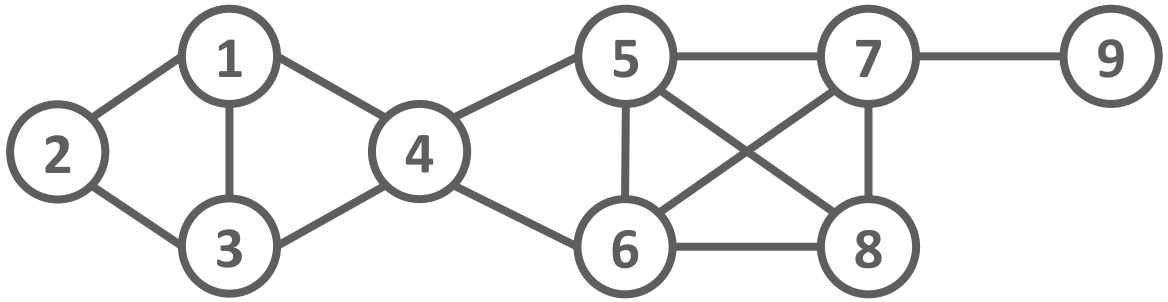
Path: sucession of vertices and edges that allow to go from one vertex to another. A path **P = (V (P), E(P))** is a subgraph such that

$$V(P) = \{v^{i0}, \ldots, v^{ik}\} \quad \text{and} \quad E(P) = \{(v^{i_0}, v^{i_1}), \ldots, (v^{i_{k-1}}, v^{i_k})\}$$

Geodesic distance: number of edges of the shortest path between two vertices

Example: in the graph above, what is the geodesic between vertices 2 and 5 ?

Consider the following graph **G = (V, E)** where **|V| = 9** and **|E| = 14**



Path: sucession of vertices and edges that allow to go from one vertex to another. A path **P = (V (P), E(P))** is a subgraph such that

$$V(P) = \{v^{i0}, \ldots, v^{ik}\} \quad \text{and} \quad E(P) = \{(v^{i_0}, v^{i_1}), \ldots, (v^{i_{k-1}}, v^{i_k})\}$$

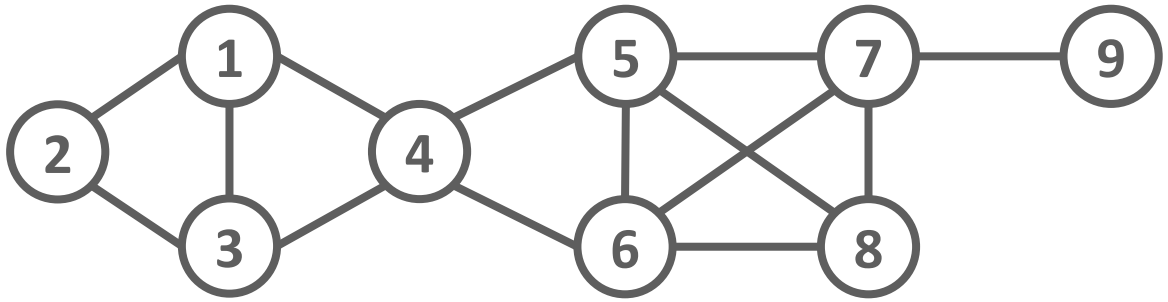
Geodesic distance: number of edges of the shortest path between two vertices

Example: in the graph above, what is the geodesic between vertices 2 and 5 ?

Connected graph: a graph is connected if, and only if, for every pair of vertices ***u*** and ***v*** there exists at least one path connecting them.

Example: is the graph above connected ?

Consider the following graph **G = (V, E)** where **|V| = 9** and **|E| = 14**



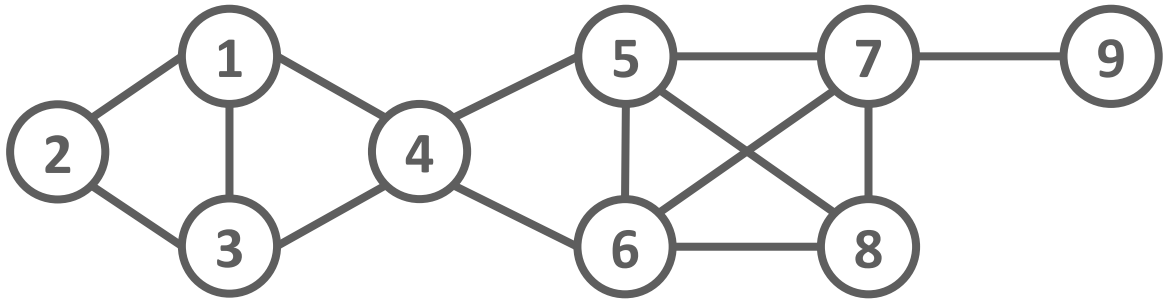
Density of a graph: ratio between the number of edges of the graph and the maximum possible number of edges between its vertices.

$$\text{density} = \frac{2 \times |E|}{|V| \times (|V| - 1)}$$

Diameter of a graph: the greatest geodesic distance possible between two vertices of the graph.

Neighbourhood of a vertex: the neighbourhood N_i is a subgraph induced by all of the vertices adjacent (i.e. connected to) vertex v_i .

Consider the following graph **G = (V, E)** where **|V| = 9** and **|E| = 14**

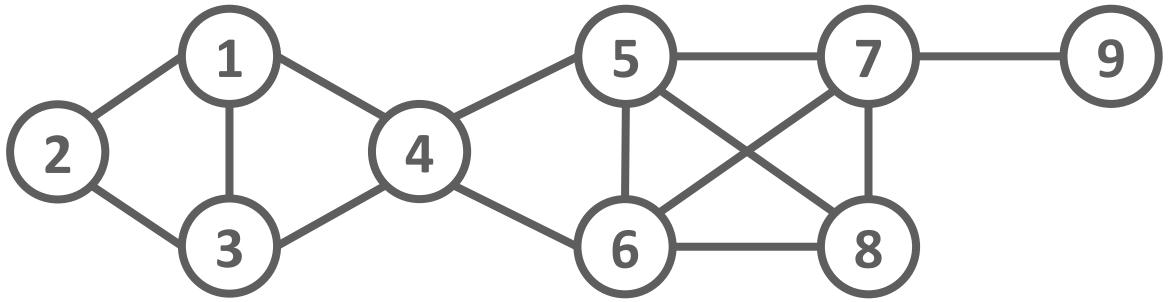


Strength of connection: the strength of connection between two nodes can be evaluated via the overlap of their neighbourhoods. The larger the intersection, the stronger the link.

$$\text{overlap}(v_i, v_j) = \frac{|N_i \cap N_j|}{|N_i \cup N_j| - 2}$$

Example: for the graph above, what is the strength of connection between v_1 and v_4 ? And between v_4 and v_5 ?

Consider the following graph **G = (V, E)** where **|V| = 9** and **|E| = 14**



Strength of connection: the strength of connection between two nodes can be evaluated via the overlap of their neighbourhoods. The larger the intersection, the stronger the link.

$$\text{overlap}(v_i, v_j) = \frac{|N_i \cap N_j|}{|N_i \cup N_j| - 2}$$

Example: for the graph above, what is the strength of connection between v_1 and v_4 ? And between v_4 and v_5 ?

Degree centrality: number of edges associated to a node. It can be absolute or relative.

$$C_D(v_i) = d_i = \sum_j A_{ij} \quad \text{and} \quad C_D^0(v_i) = \frac{C(v_i)}{n-1}$$

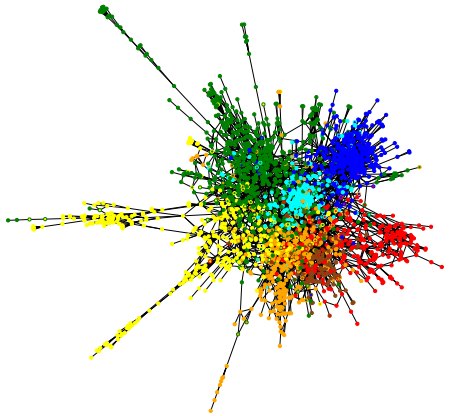
And many other possible definitions…

Graph signal processing: Fourier basis of a graph, filters over graphs, sub-sampling

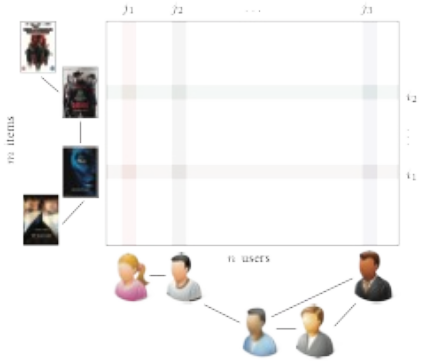
Ortega et al. “Graph Signal Processing: Overview, Challenges and Applications” (2018)
arxiv.org/pdf/1712.00468.pdf

Geometric deep learning: Neural networks over graphs

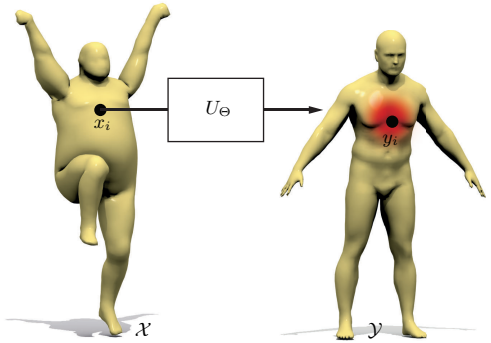
Bronstein et al. “Geometric deep learning: going beyond Euclidean data”
arxiv.org/pdf/1611.08097.pdf



Semi-supervised classification



Recommendation systems



Shape correspondence analysis

Part 1: Graphs basics

Part 2: Community detection

What is a community?

Community (social): group of people who have special ties because they have particular affinities, or have similar characteristics, or share interests.

What is a community?

Community (social): group of people who have special ties because they have particular affinities, or have similar characteristics, or share interests.

Community (graph): a set of nodes that are strongly linked to each other, and weakly linked with nodes outside of the community.

$\Longrightarrow$ This implies that it must exist at least a path between two nodes of a community, and this path must be internal to the community.

What is a community?

Community (social): group of people who have special ties because they have particular affinities, or have similar characteristics, or share interests.

Community (graph): a set of nodes that are strongly linked to each other, and weakly linked with nodes outside of the community.

$\Longrightarrow$ This implies that it must exist at least a path between two nodes of a community, and this path must be internal to the community.

Clustering: in the clustering framework, a community is a cluster of nodes.

Community (social): group of people who have special ties because they have particular affinities, or have similar characteristics, or share interests.

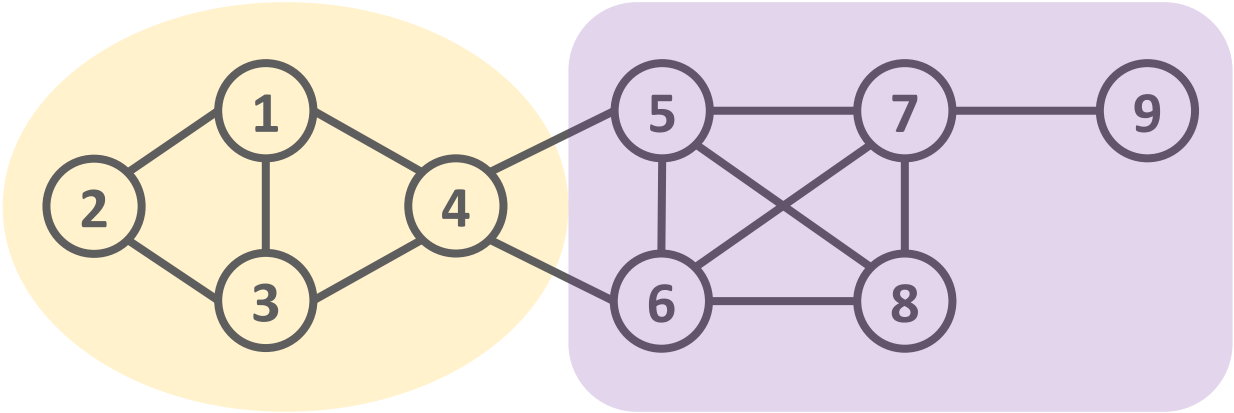
Community (graph): a set of nodes that are strongly linked to each other, and weakly linked with nodes outside of the community.

$\Longrightarrow$ This implies that it must exist at least a path between two nodes of a community, and this path must be internal to the community.

Clustering: in the clustering framework, a community is a cluster of nodes.

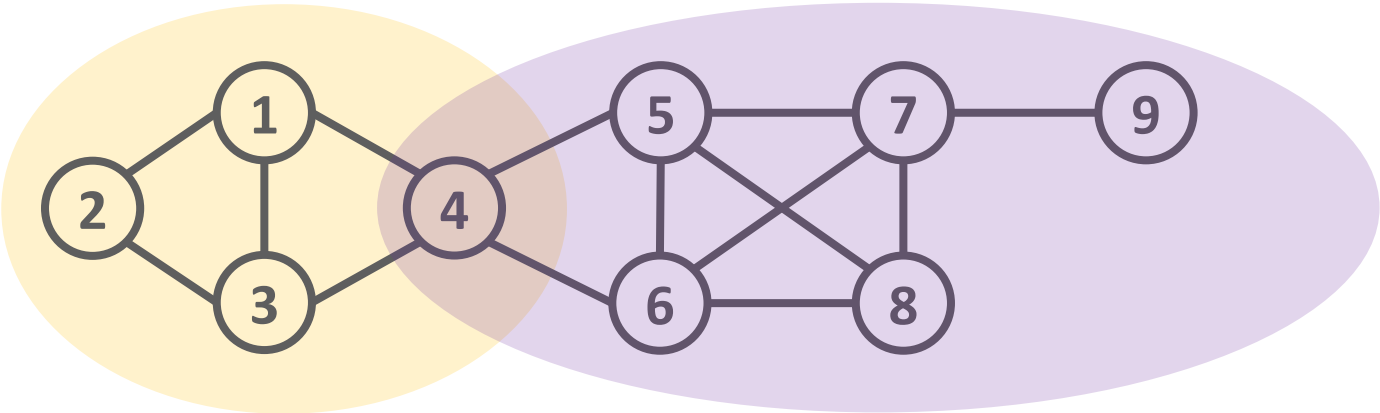
CHALLENGES

- How to **define** a community?
- How **many** communities are there?
- How evaluate the "**quality**" of the split?



Disjoint versus overlapping communities

Just like in clustering, the partition of the nodes can be **crisp** (disjoint communities) or **fuzzy** (overlapping communities). The membership of a vertex to a community can sometimes be **ambiguous**.



Example: in the graph above, the node v_4 belongs to two communities, acting as an interface between them. Notions of **centrality** and **influence** of a node become relevant in this situation.

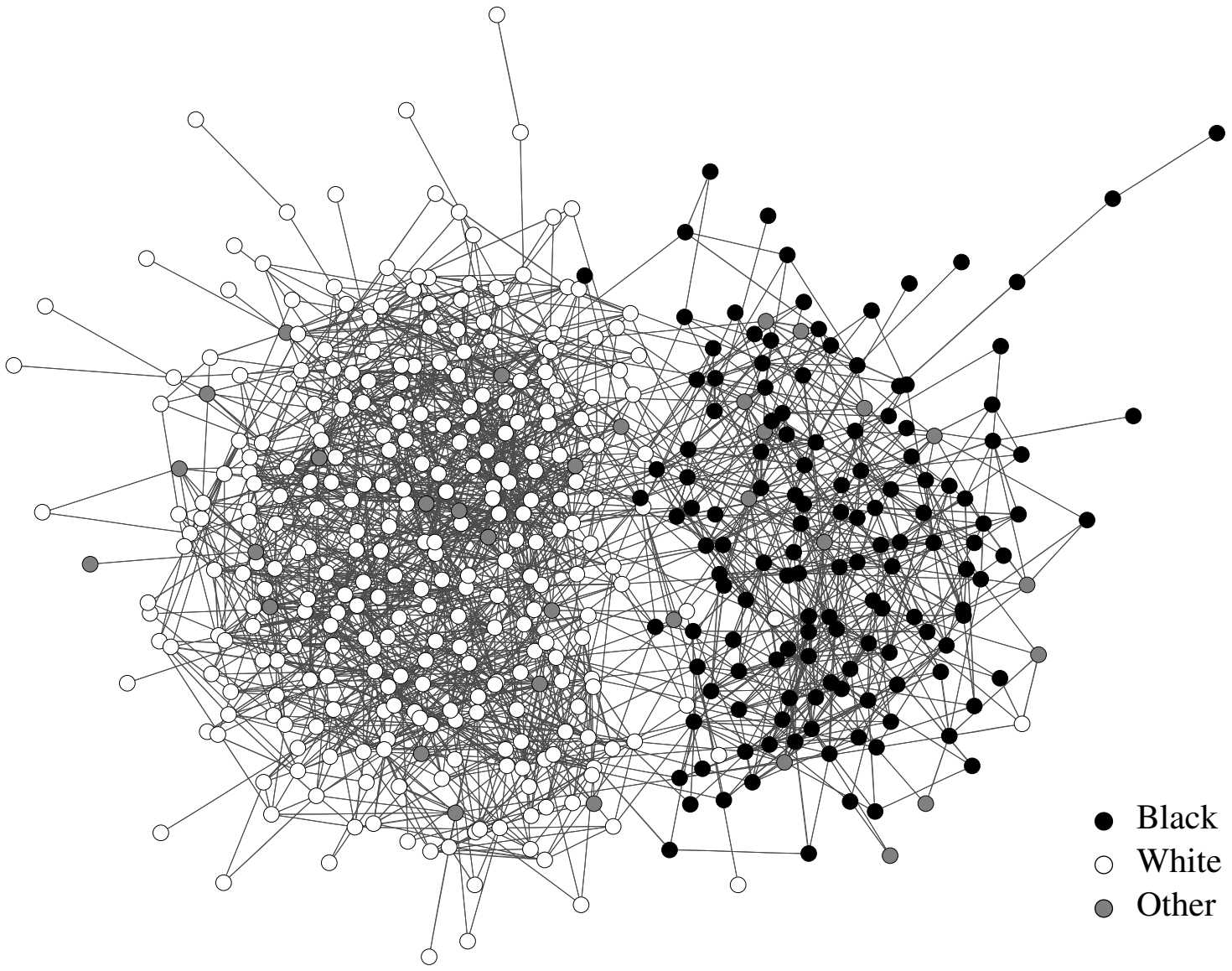
For the rest of this course, we will **consider only disjoint** communities.

Friendship graph in a US High School

The 470 nodes represent students at a US high school. The nodes are color coded by race as indicated in the legend.

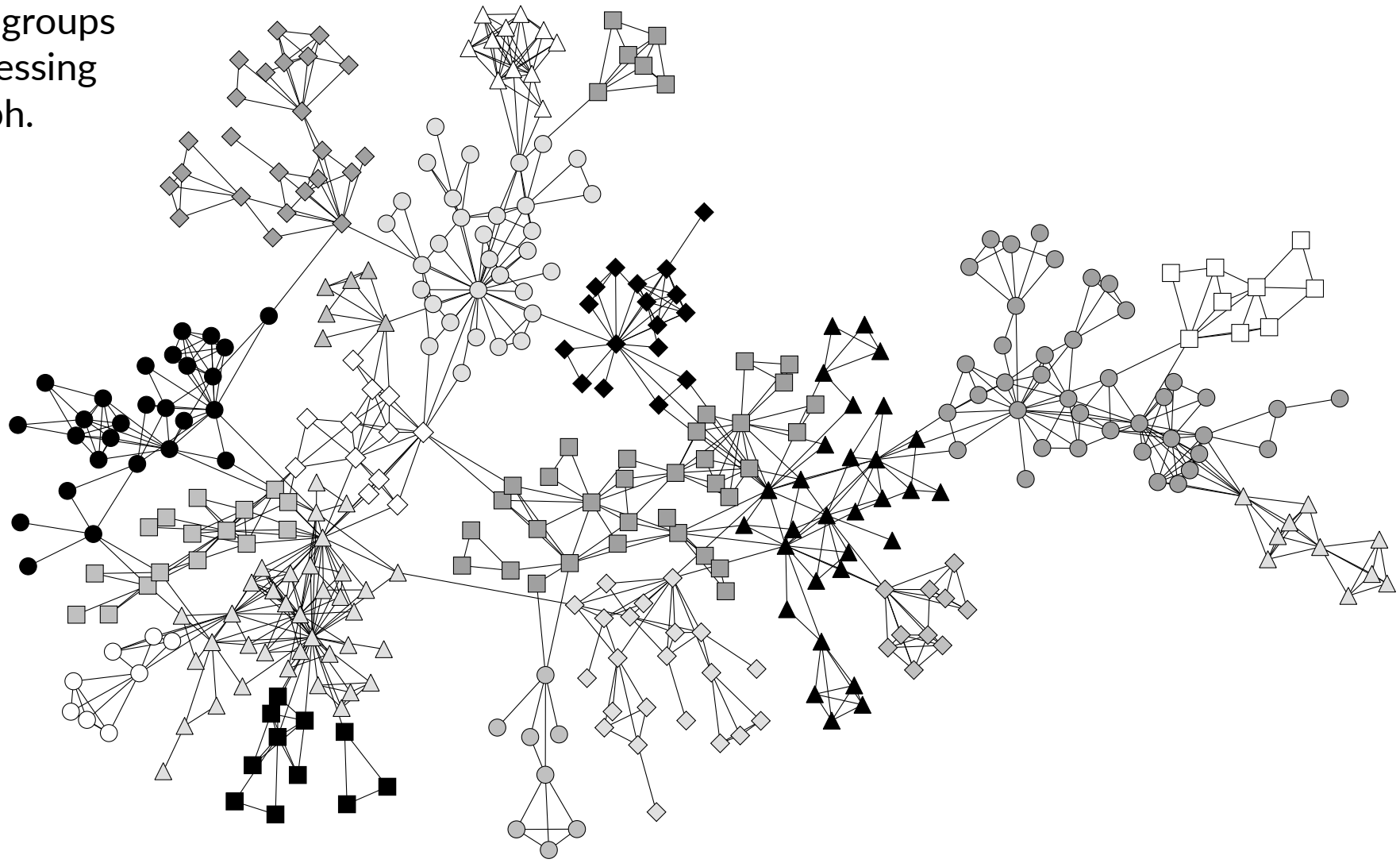
Each student was presented to photos of 10 random other students from the same school and asked which of them he/she considered as a friend. The edges represent the positive answers to that question.

Community detection in this situation can help better understanding the social interactions in a public school and evaluate which actions can be made for better integration.



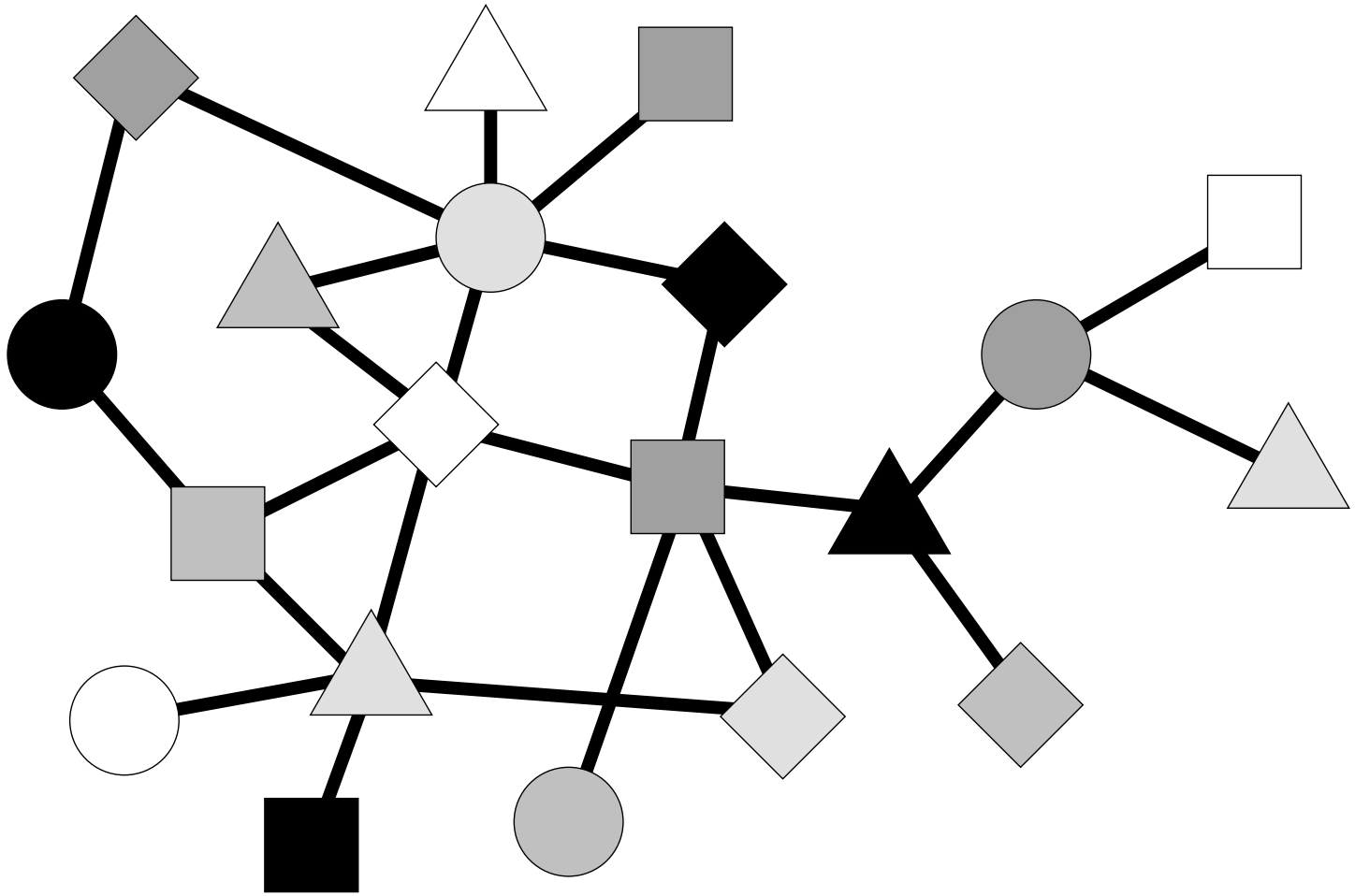
Multi-scale analysis of large graphs

In large-scale graphs such as this one, it is often very useful to represent groups of nodes so to alleviate the processing times for the analysis of the graph.

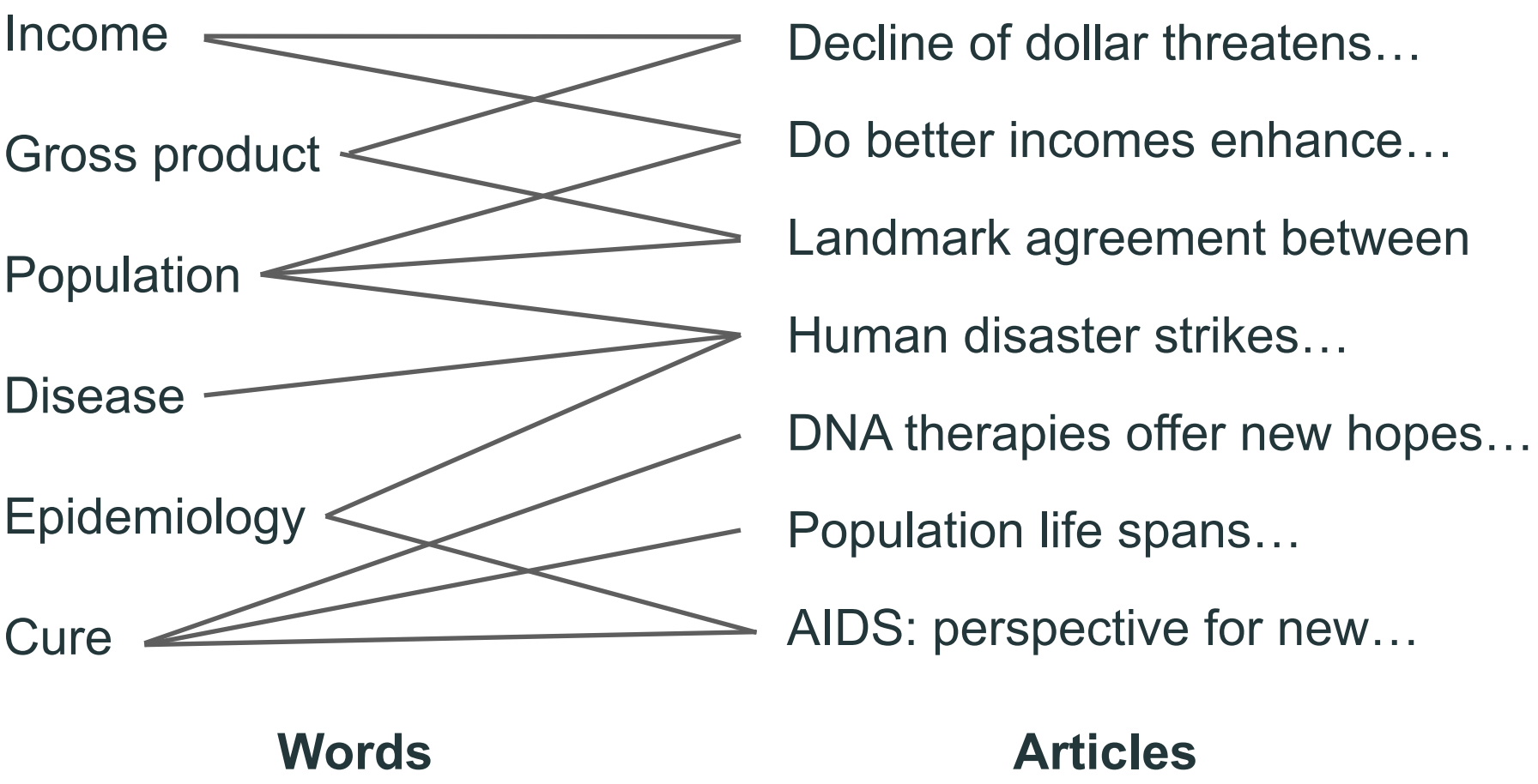


Multi-scale analysis of large graphs

In large-scale graphs such as this one, it is often very useful to represent groups of nodes so to alleviate the processing times for the analysis of the graph.

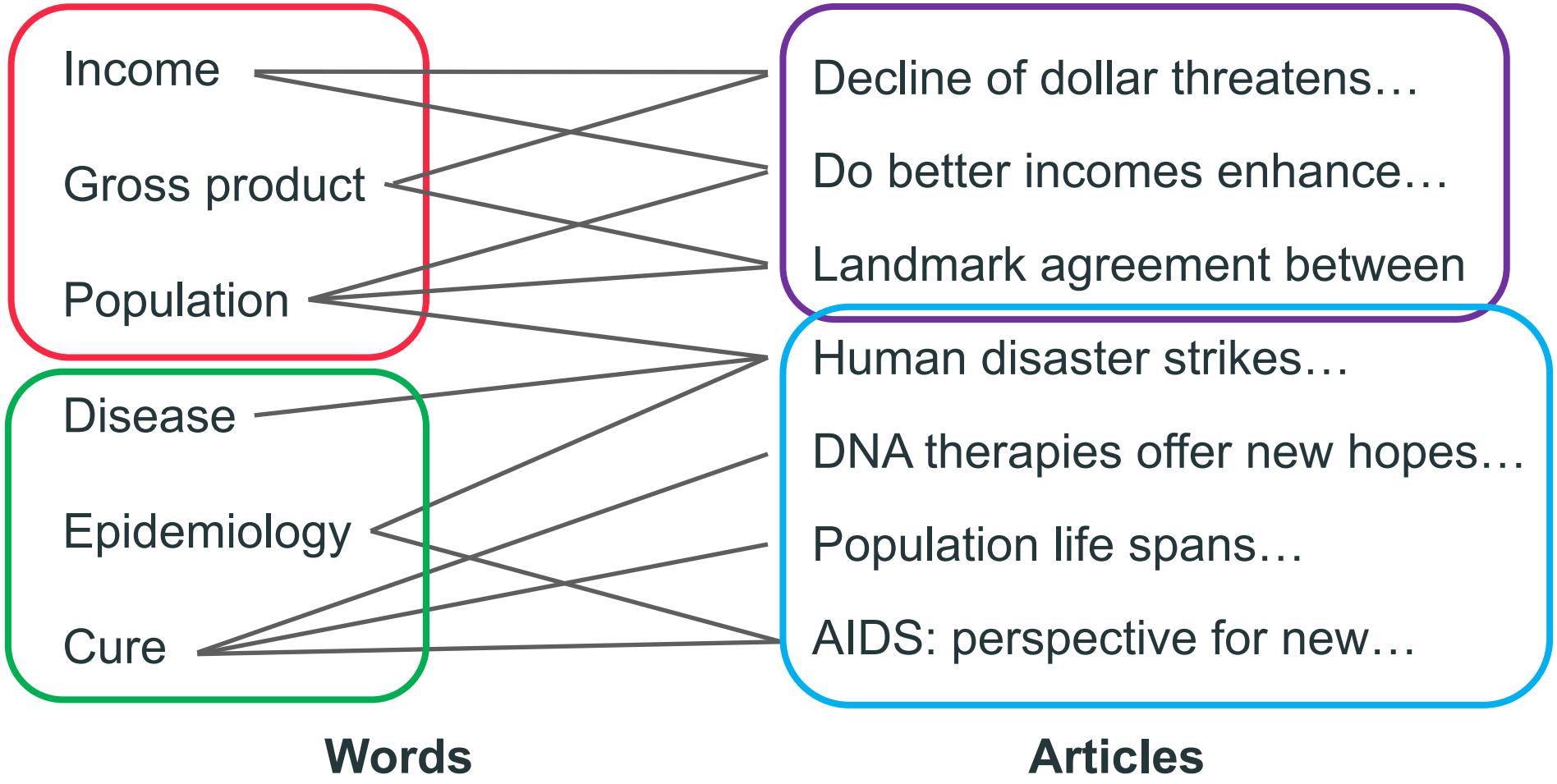


How to apply these concepts to document mining?



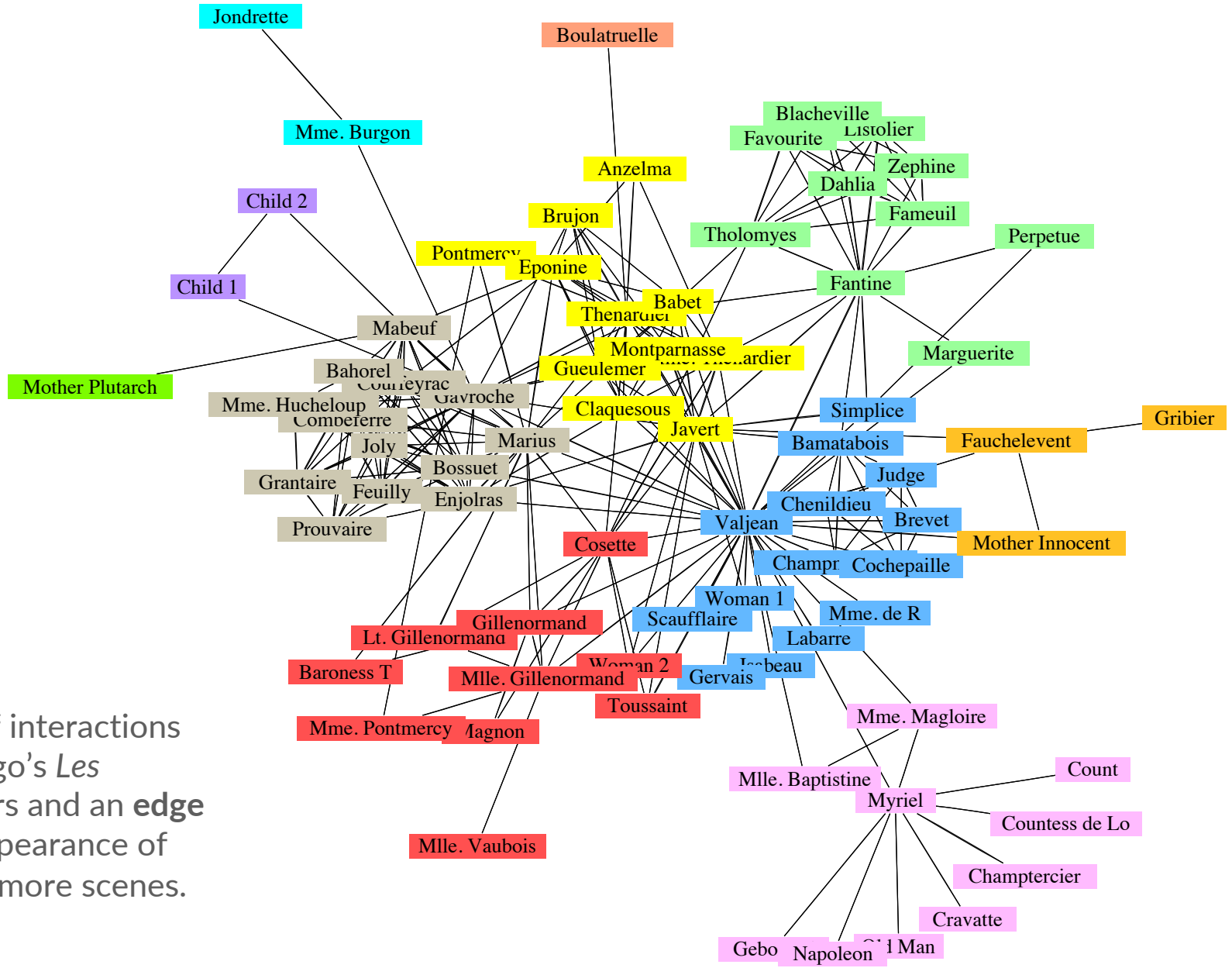
Clustering of documents sharing the same words / topics

How to apply these concepts to document mining?



Clustering of documents sharing the same words / topics

How to apply these concepts to document mining?



Community structure of the network of interactions between major characters in Victor Hugo's *Les Misérables*. **Vertices** represent characters and an **edge** between two vertices represents co-appearance of the corresponding characters in one or more scenes.

Part 2: Community detection

We are going to study four different methods for community detection in graphs.

- Hierarchical clustering
  - Agglomerative
  - Divisive
- Spectral community detection
- Louvain algorithm

Part 2: Community detection

We are going to study four different methods for community detection in graphs.

- Hierarchical clustering
  - Agglomerative **[similarity / dissimilarity]**
  - Divisive **[edge betweenness]**
- Spectral community detection **[modularity]**
- Louvain algorithm **[modularity]**

But first… why not **K-means** ?

In K-means, we seek to partition the data points into a **pre-specified** number of clusters.

The algorithm (K-means)

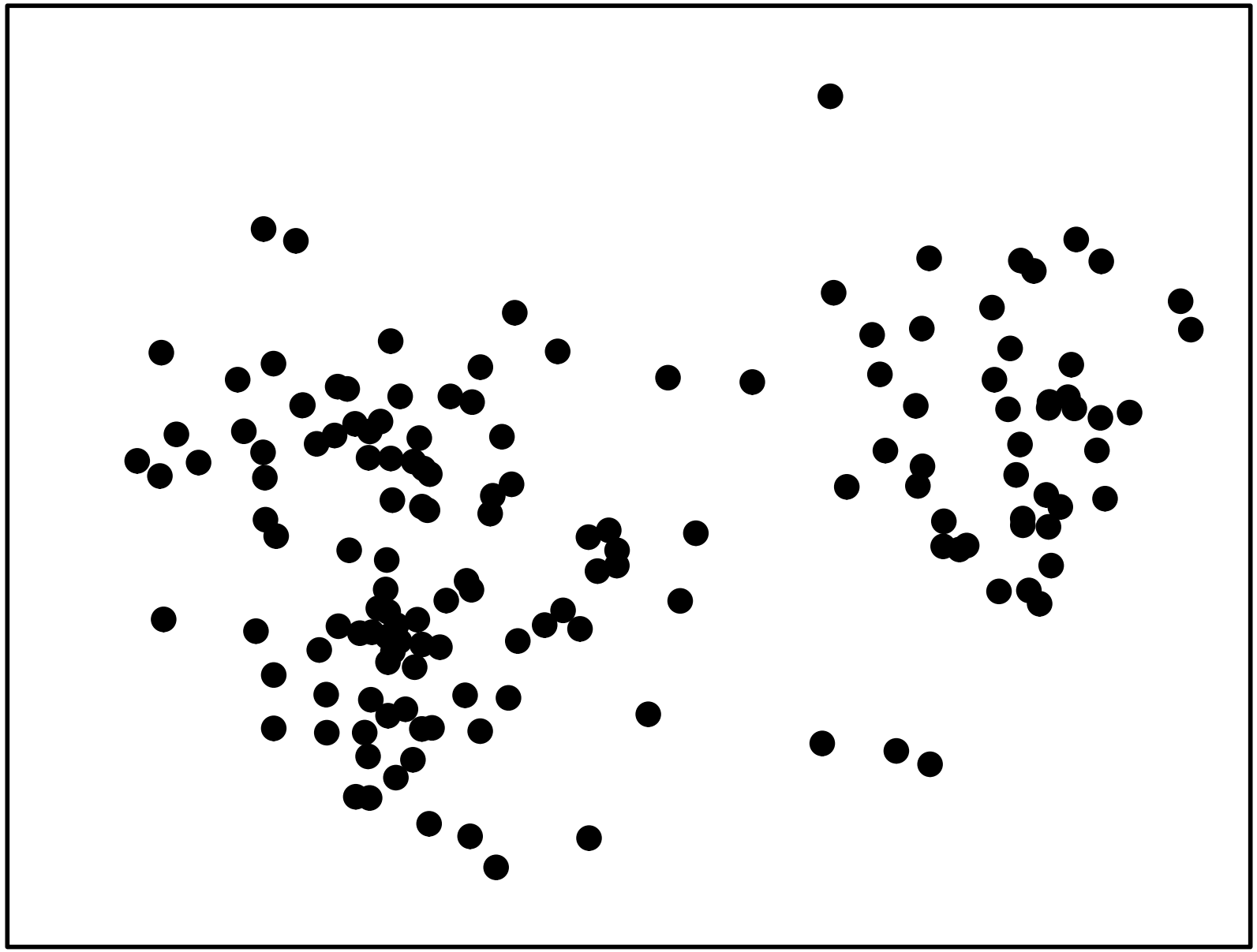
1) Randomly assign a number, from 1 to K, to each of the data points. These serve as initial cluster assignments.

2) Iterate until the cluster assignments stop changing:

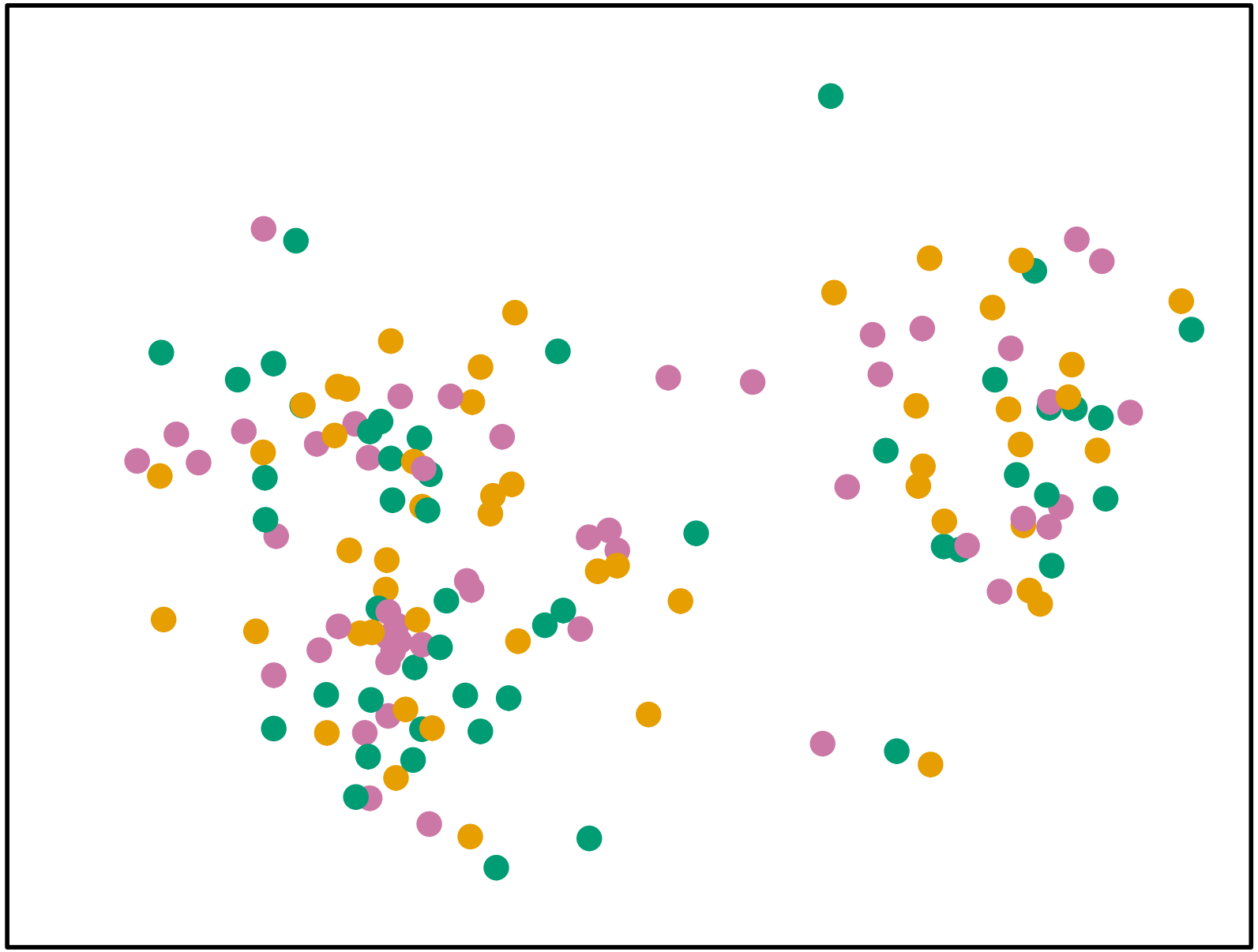
    1) For each of the K clusters, compute the cluster **centroid**.

    2) Assign each observation to the cluster whose centroid is **closest**.

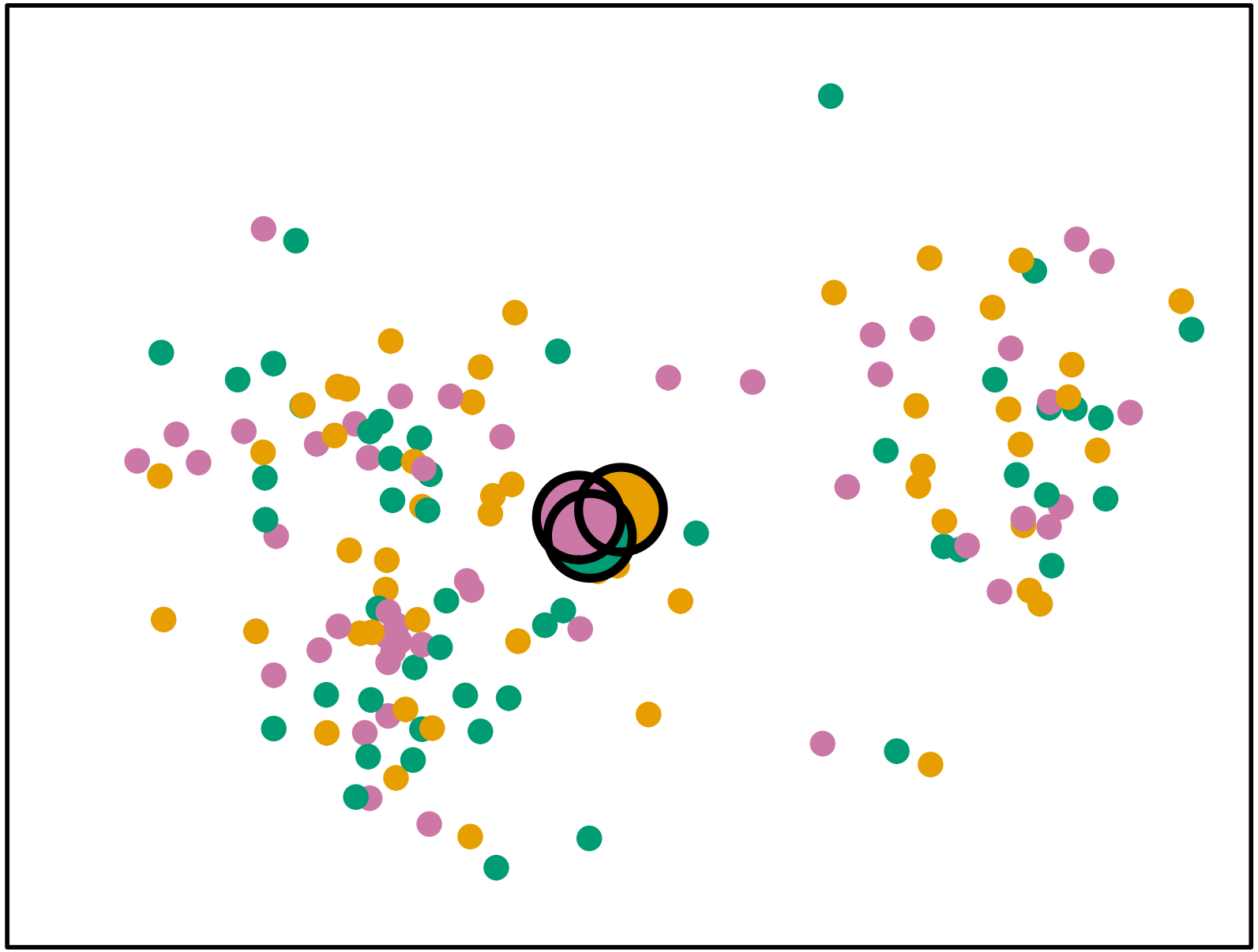
Initial dataset



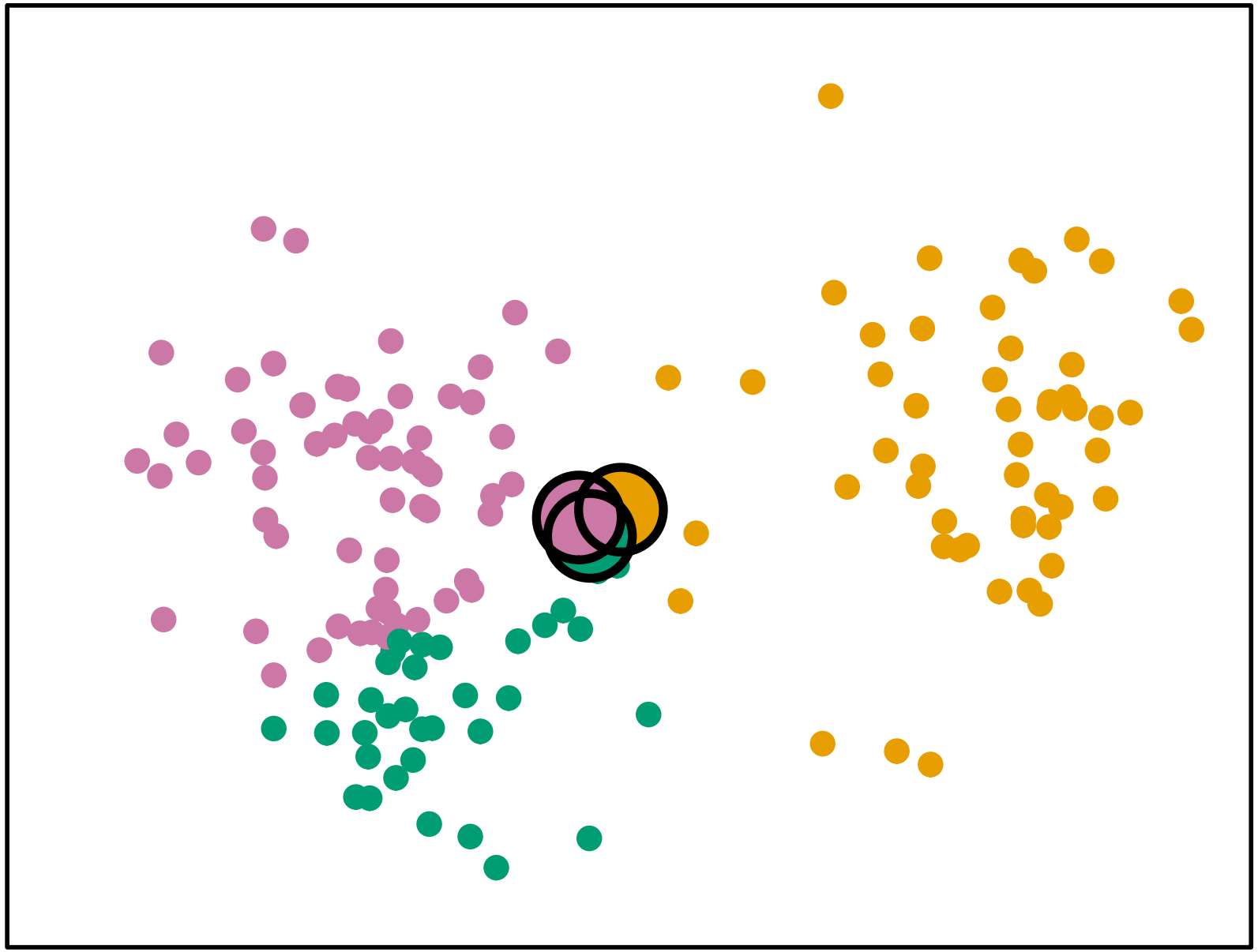
Step 1



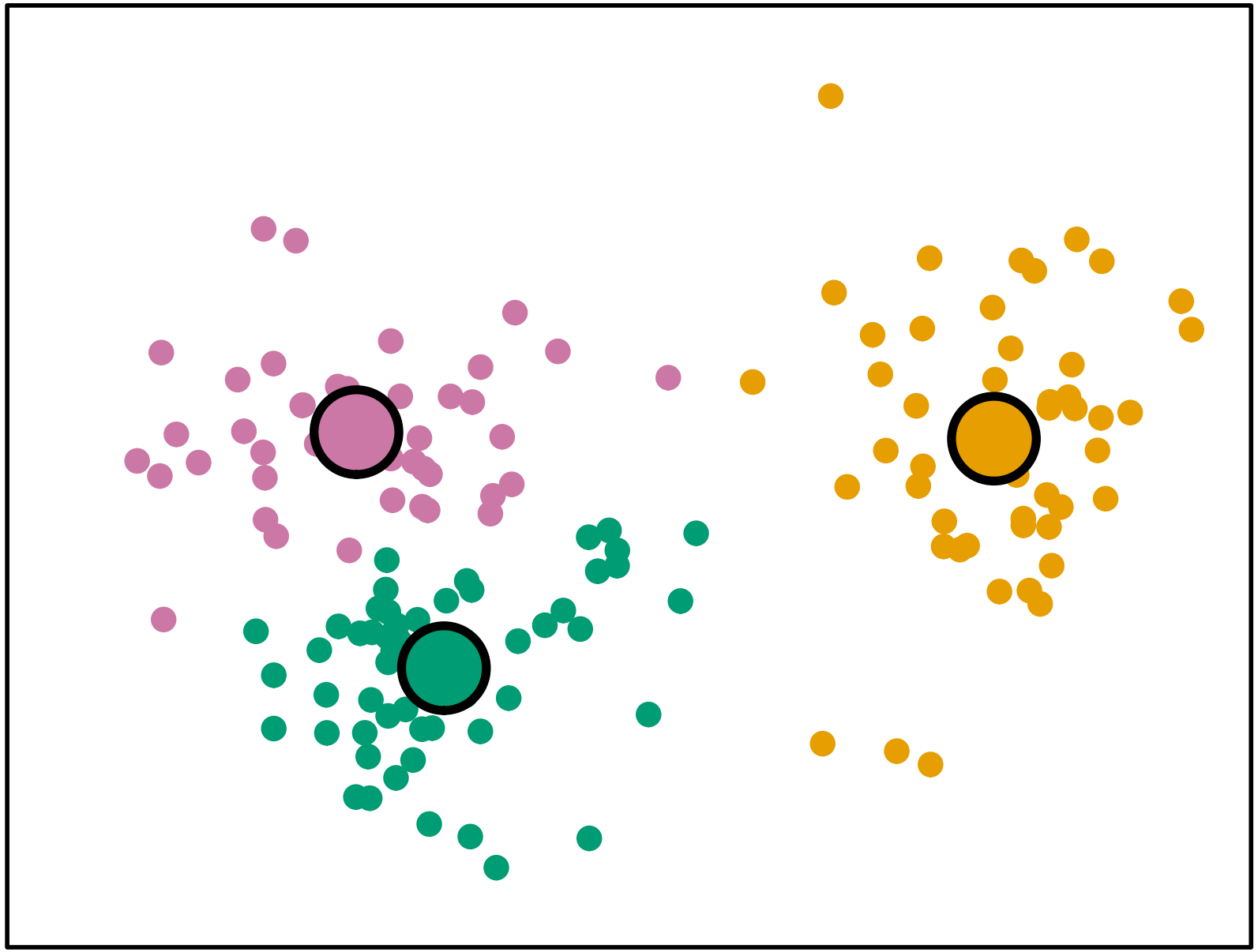
Step 2.1 – Iteration 1



Step 2.2 – Iteration 1



Step 2.2 – Iteration 1



Can we use this procedure for community detection in **graphs**?

Yes… but it would be **hard**. No closed formula for **centroid** or closest **distance**

The algorithm (K-means)

1) Randomly assign a number, from 1 to K, to each of the data points. These serve as initial cluster assignments.

2) Iterate until the cluster assignments stop changing:

    1) For each of the K clusters, compute the cluster **centroid**.

    2) Assign each observation to the cluster whose centroid is **closest**.

These quantities require a metric space, which we don't always have for data represented in a graph.

Part 2: Community detection

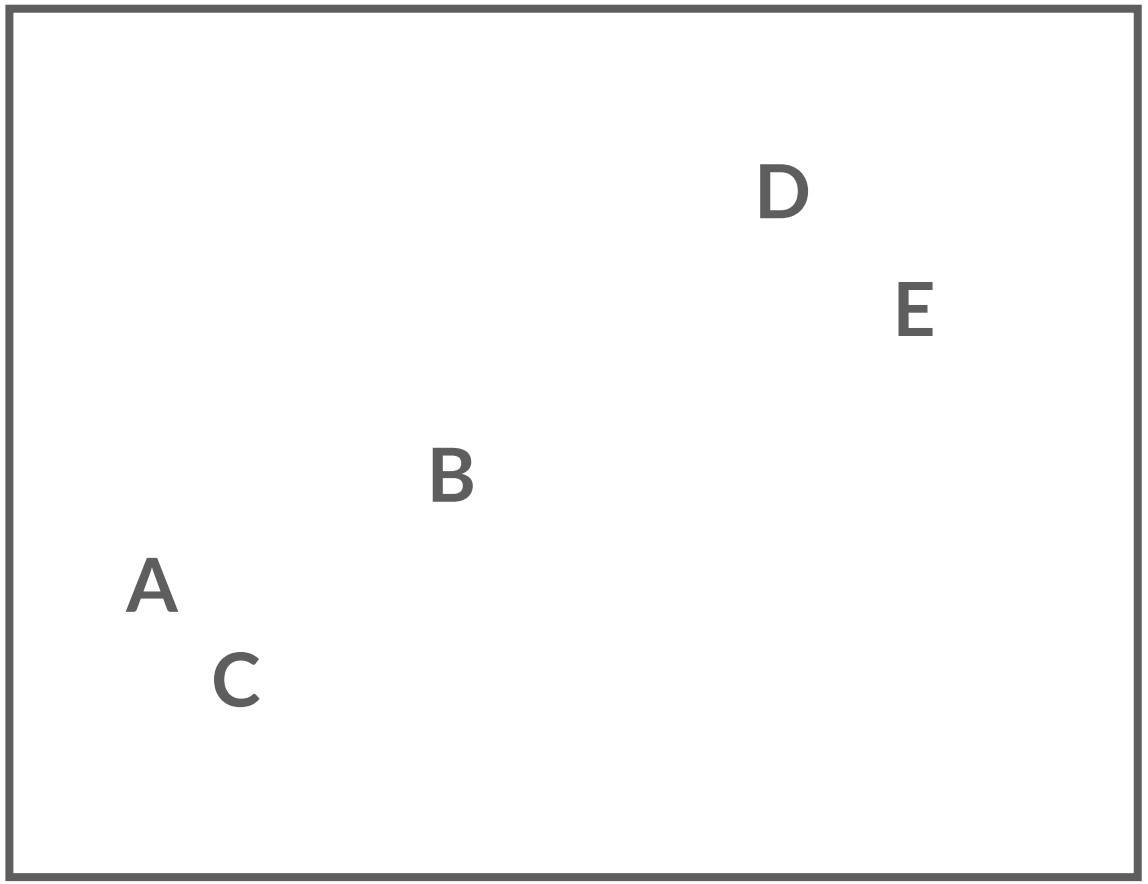
We are going to study four different methods for community detection in graphs.

- Hierarchical clustering
  - Agglomerative **[similarity / dissimilarity]**
  - Divisive **[edge betweenness]**
- Spectral community detection **[modularity]**
- Louvain algorithm **[modularity]**

Hierarchical clustering: agglomerative procedure

K-means clustering requires us to **pre-specify** the number of clusters, which can be hard.

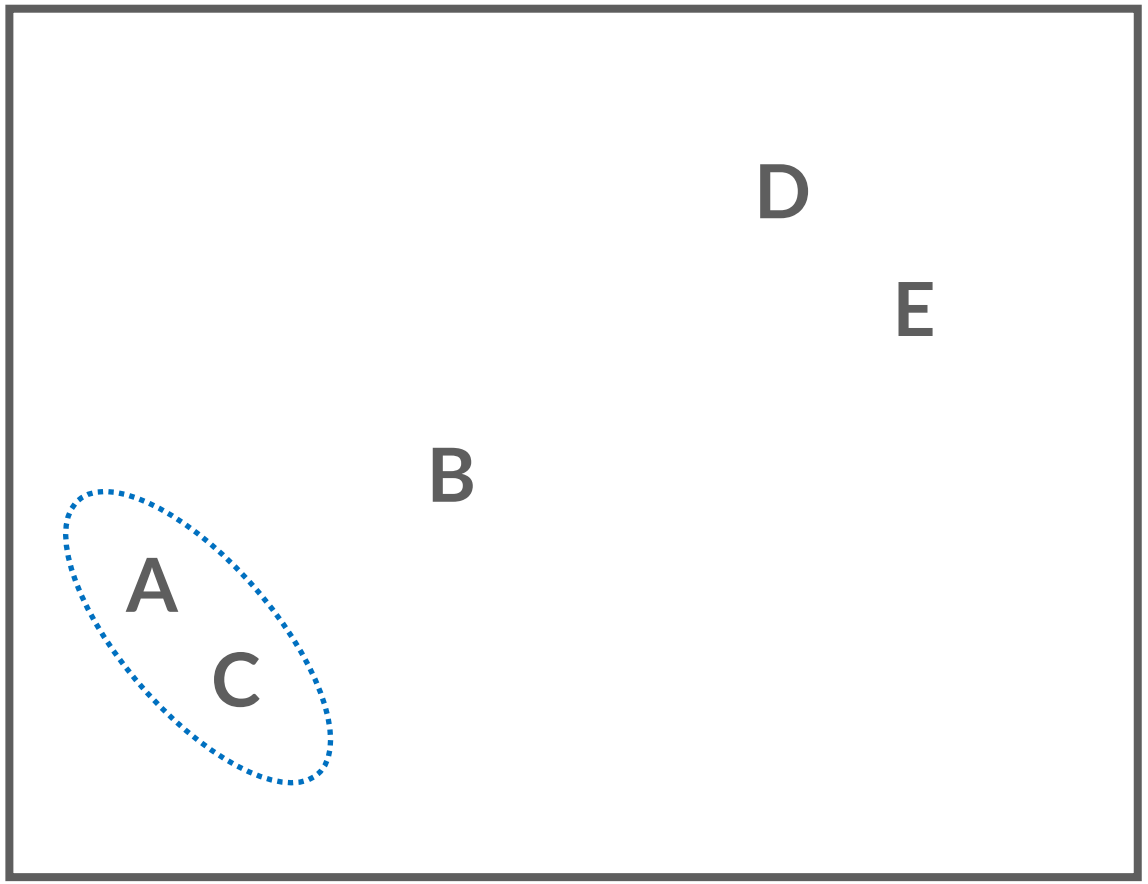
In hierarchical clustering, we provide an object showing all possible clusters of a dataset.



Hierarchical clustering: agglomerative procedure

K-means clustering requires us to **pre-specify** the number of clusters, which can be hard.

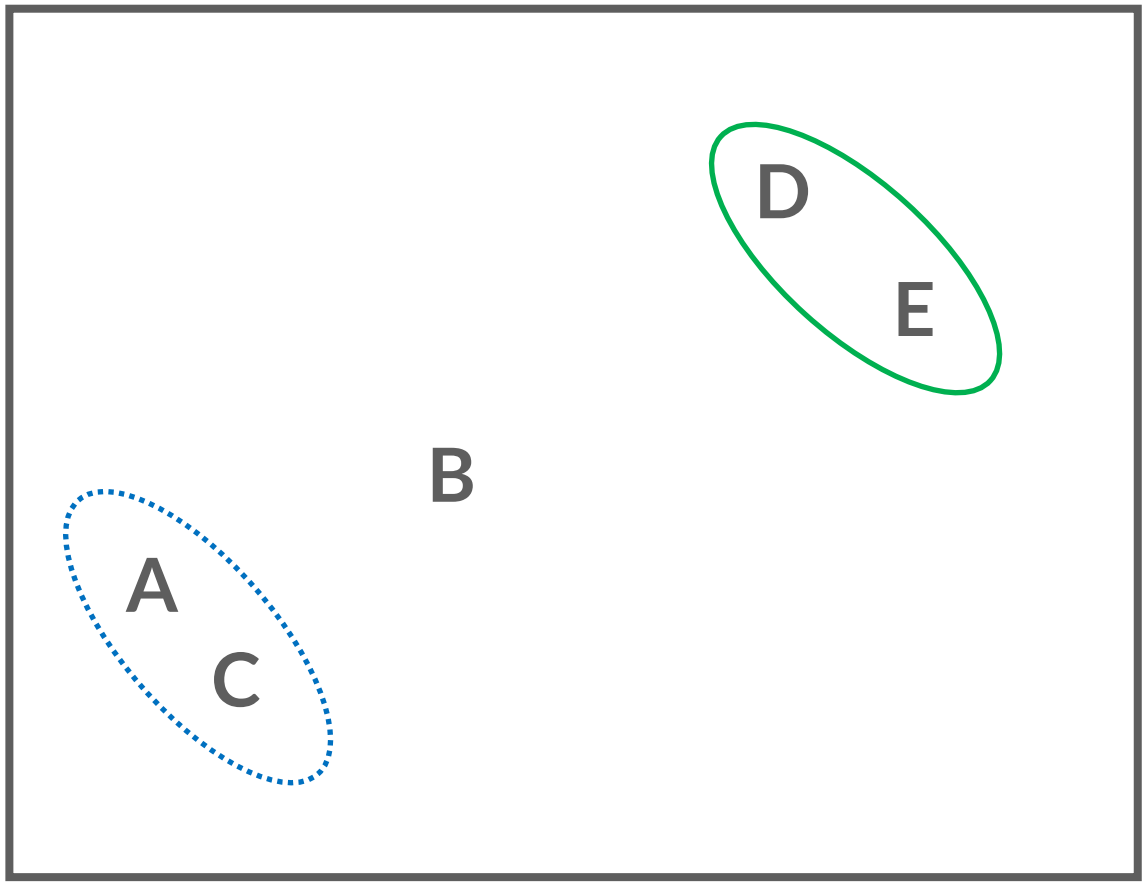
In hierarchical clustering, we provide an object showing all possible clusters of a dataset.



Hierarchical clustering: agglomerative procedure

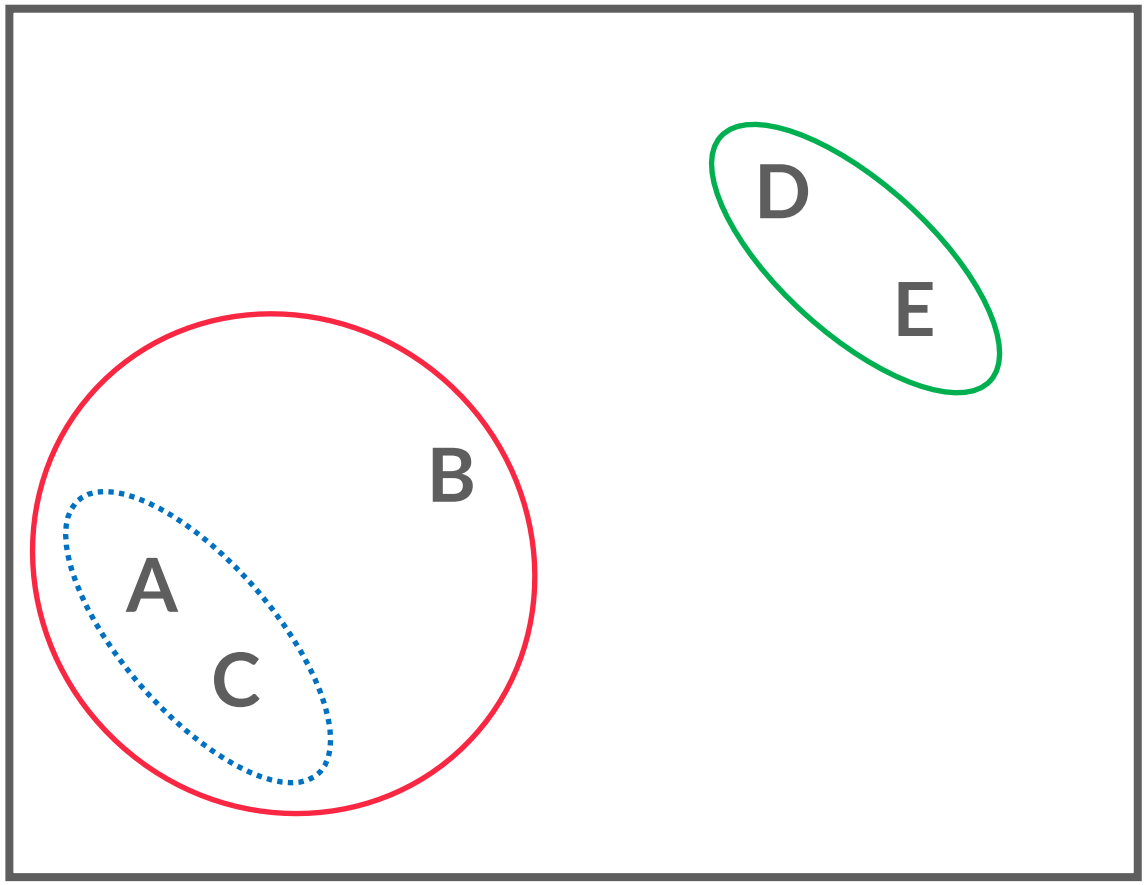
K-means clustering requires us to **pre-specify** the number of clusters, which can be hard.

In hierarchical clustering, we provide an object showing all possible clusters of a dataset.



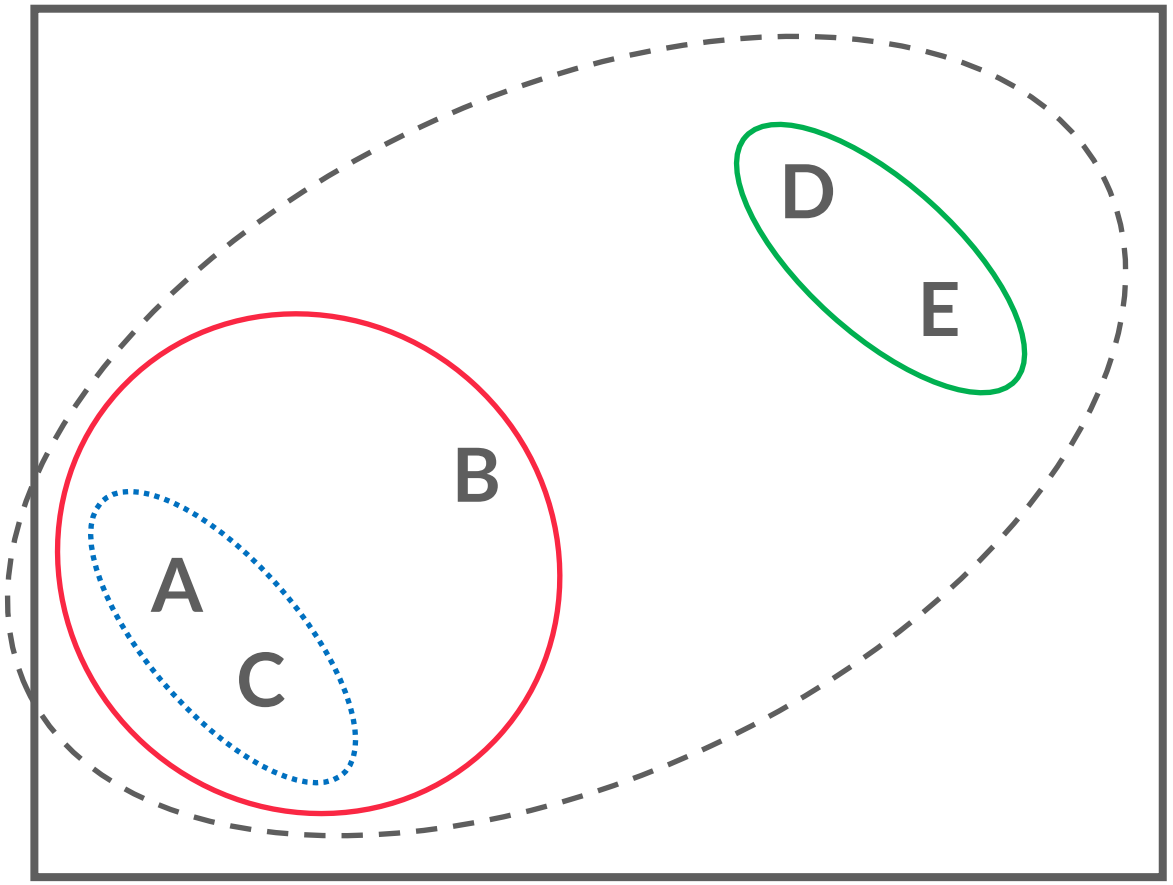
K-means clustering requires us to **pre-specify** the number of clusters, which can be hard.

In hierarchical clustering, we provide an object showing all possible clusters of a dataset.



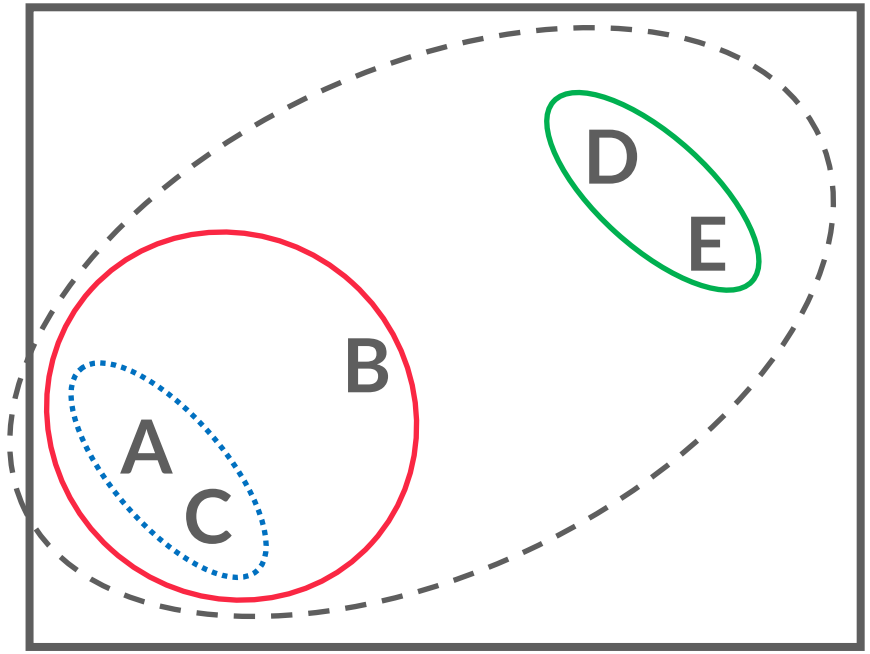
K-means clustering requires us to **pre-specify** the number of clusters, which can be hard.

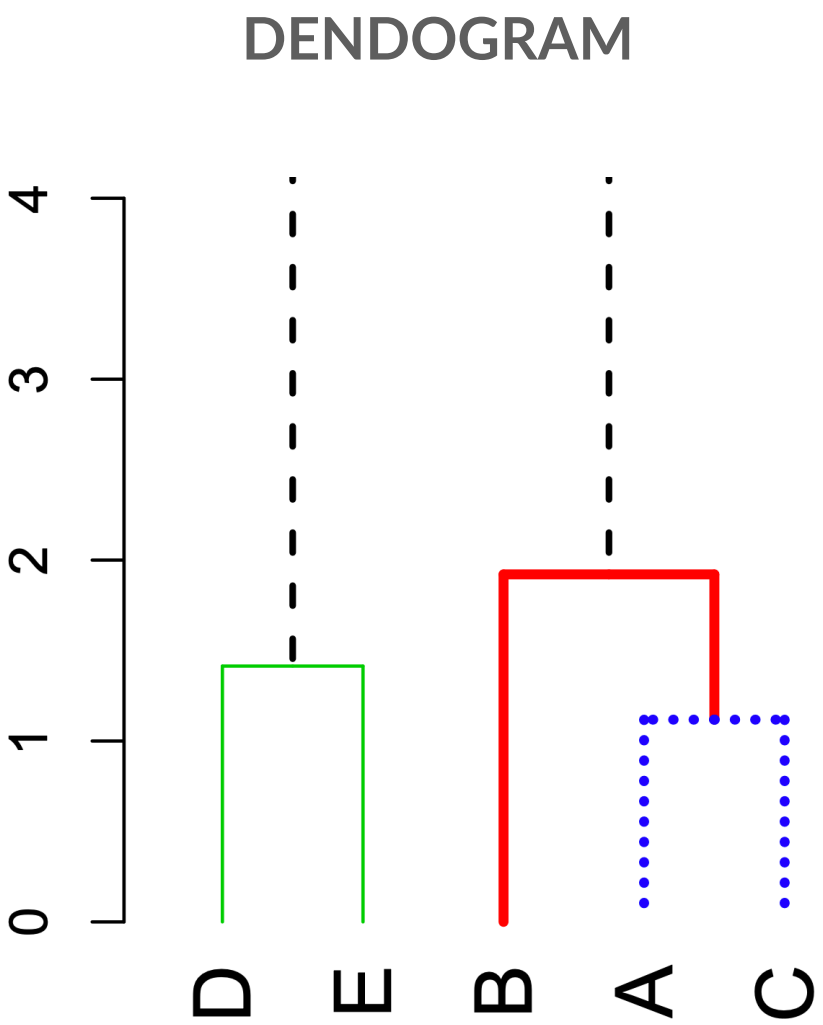
In hierarchical clustering, we provide an object showing all possible clusters of a dataset.



The algorithm (Agglomerative hierarhical clustering)

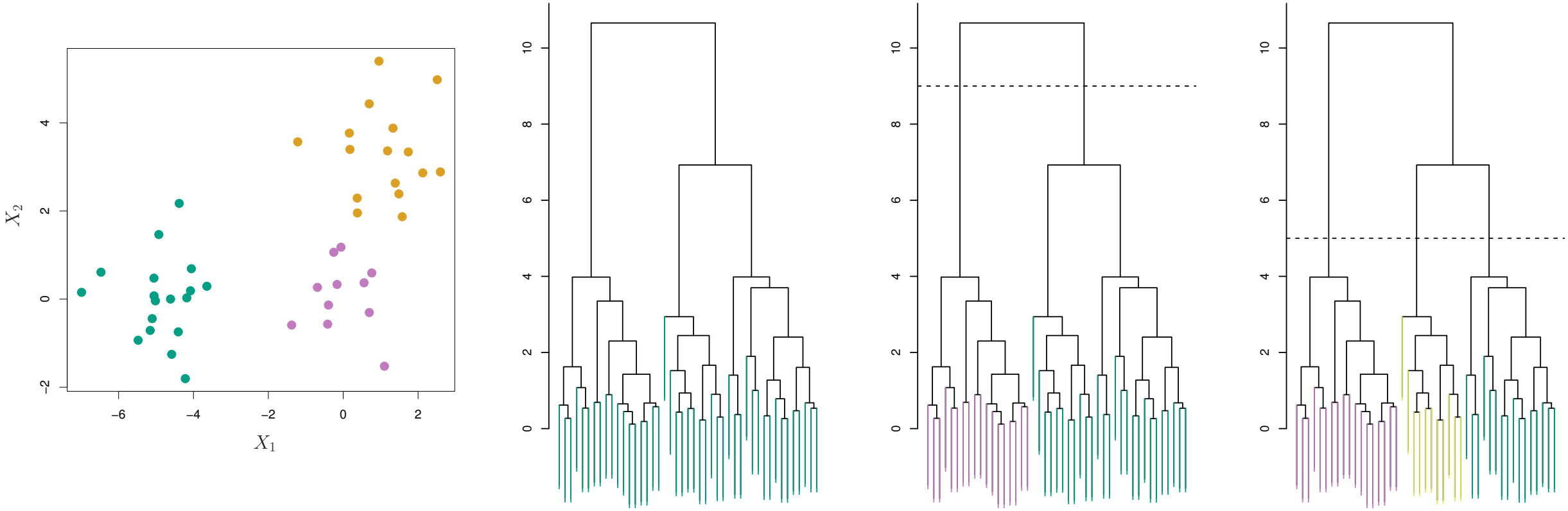
1) Start with each point being its own cluster
2) Identify the **closest** two clusters and merge them
3) Repeat
4) End when all points are in a single cluster





Hierarchical clustering: agglomerative procedure

Example: we have data disposed as in the figure. The colors are the true clusters and we apply hiearchical clustering without using this information.



The algorithm (Agglomerative hierarhical clustering)

1) Start with each point being its own cluster
2) Identify the **closest** two clusters and merge them
3) Repeat
4) End when all points are in a single cluster

Two **important questions**:

- How to define the distance between two vertices of a graph?
- How to define the distance between two clusters?

- How to define the distance between two vertices of a graph?

Similarity: In graph theory, the similarity is defined in terms of neighbourhood. Two nodes are close if there is a large overlap between their neighbourhoods.

Two examples of measure of such similarity are the **Jaccard** index

$$\text{Jaccard}(v_i, v_j) = \frac{|N_i \cap N_j|}{|N_i \cap N_j|}$$

And the **cosine** similarity

$$\text{Cosine}(v_i, v_j) = \frac{|N_i \cap N_j|}{\sqrt{|N_i| \times |N_j|}}$$

Any decreasing function of the similarity can be seen as a **distance** between vertices.

- How to define the distance between two clusters?

For a given dissimilarity measure *dis* between two vertices, several measures *D* of dissimilarity between clusters exists:

Single linkage: Minimal inter-cluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B and record the **smallest** of these dissimilarities

$$D(A, B) = \min\{\mathrm{dis}(x, y) : x \in A, y \in B\}$$

Complete linkage: Maximal inter-cluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B and record the **largest** of these dissimilarities.

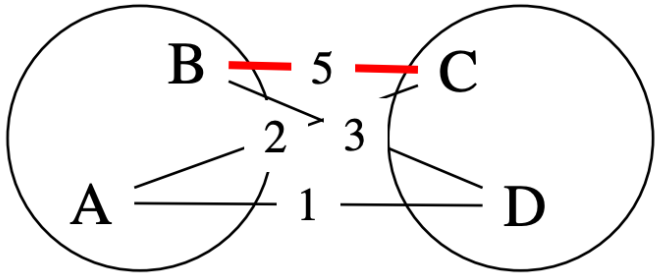
$$D(A, B) = \max\{\mathrm{dis}(x, y) : x \in A, y \in B\}$$

Average linkage: Mean inter-cluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B and record the **average** of these dissimilarities.

$$D(A, B) = \frac{1}{|A| \times |B|} \sum_{x \in A} \sum_{y \in B} \mathrm{dis}(x, y)$$

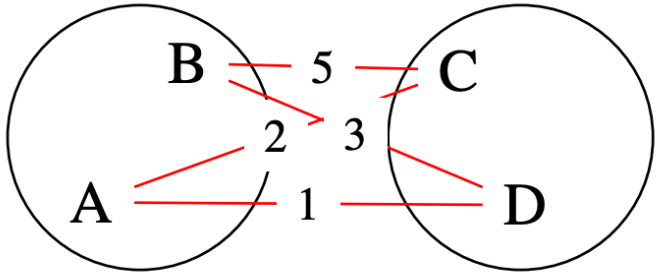
- How to define the distance between two clusters?

Single Linkage



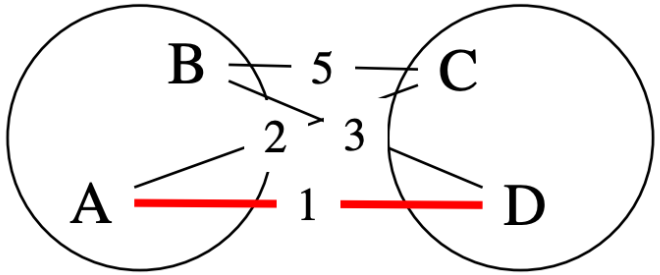
Similarity: 5

Average Linkage



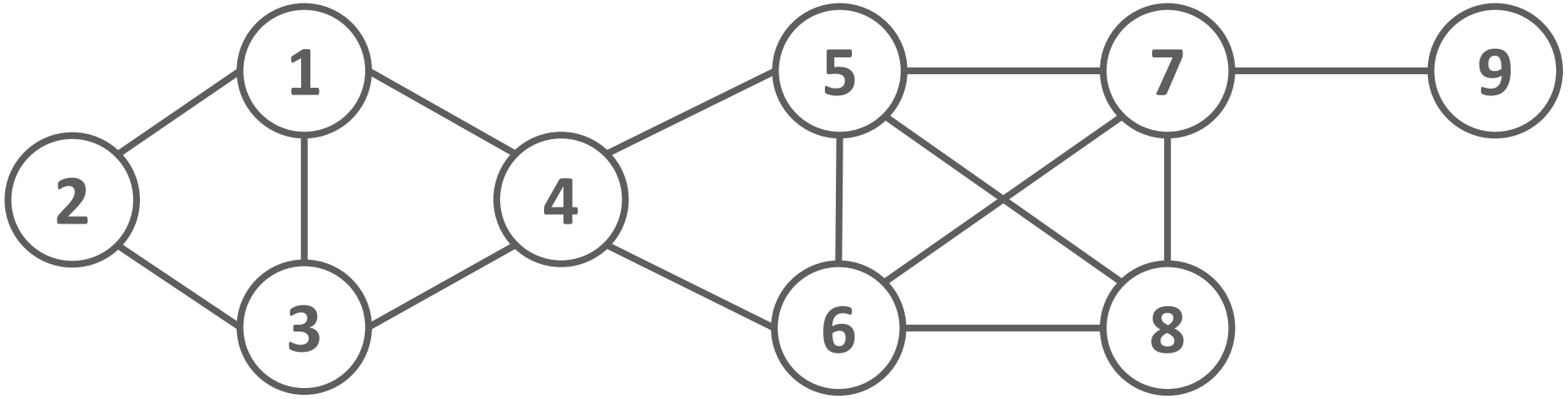
Similiarity: 2.75

Complete Linkage



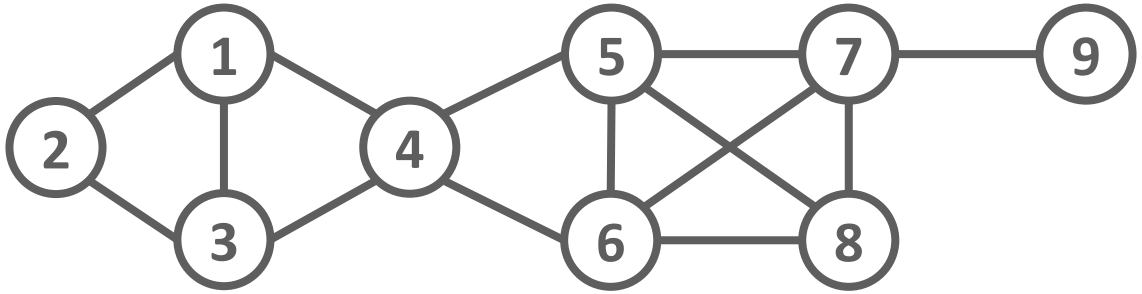
Similiarity: 1

Example: consider the graph below and do the two first iterations of hierarchical clustering using average linkage and cosine similarity.



- **Step 1**: determine the neighbourhoods for each vertex
- **Step 2**: calculate the similarity between each pair of vertices
- **Step 3**: run the hierarchical clustering algorithm

- **Step 1**: determine the neighbourhoods for each vertex



$N_1 = \{2, 3, 4\}$

$N_2 = \{1, 3\}$

$N_3 = \{1, 2, 4\}$

$N_4 = \{1, 3, 5, 6\}$

$N_5 = \{4, 6, 7, 8\}$

$N_6 = \{4, 5, 7, 8\}$

$N_7 = \{5, 6, 8, 9\}$

$N_8 = \{6, 7, 9\}$

$N_9 = \{7\}$

- **Step 2**: calculate the similarity between each pair of vertices

$N_1 = \{2, 3, 4\}$

$N_2 = \{1, 3\}$

$N_3 = \{1, 2, 4\}$

$N_4 = \{1, 3, 5, 6\}$

$N_5 = \{4, 6, 7, 8\}$

$N_6 = \{4, 5, 7, 8\}$

$N_7 = \{5, 6, 8, 9\}$

$N_8 = \{6, 7, 9\}$

$N_9 = \{7\}$

$$\text{Cosine}(v_i, v_j) = \frac{|N_i \cap N_j|}{\sqrt{|N_i| \times |N_j|}}$$

	1	2	3	4	5	6	7	8	9
1	1.00	0.41	0.67	0.29	0.29	0.29	0.00	0.00	0.00
2	0.41	1.00	0.41	0.71	0.00	0.00	0.00	0.00	0.00
3	0.67	0.41	1.00	0.29	0.29	0.29	0.00	0.00	0.00
4	0.29	0.71	0.29	1.00	0.25	0.25	0.50	0.29	0.00
5	0.29	0.00	0.29	0.25	1.00	0.75	0.50	0.58	0.50
6	0.29	0.00	0.29	0.25	0.75	1.00	0.50	0.29	0.50
7	0.00	0.00	0.00	0.50	0.50	0.50	1.00	0.58	0.00
8	0.00	0.00	0.00	0.29	0.58	0.29	0.58	1.00	0.58
9	0.00	0.00	0.00	0.00	0.50	0.50	0.00	0.58	1.00

- **Step 3**: run the hierarchical clustering algorithm

	1	2	3	4	5	6	7	8	9
1	1.00	0.41	0.67	0.29	0.29	0.29	0.00	0.00	0.00
2	0.41	1.00	0.41	0.71	0.00	0.00	0.00	0.00	0.00
3	0.67	0.41	1.00	0.29	0.29	0.29	0.00	0.00	0.00
4	0.29	0.71	0.29	1.00	0.25	0.25	0.50	0.29	0.00
5	0.29	0.00	0.29	0.25	1.00	0.75	0.50	0.58	0.50
6	0.29	0.00	0.29	0.25	0.75	1.00	0.50	0.29	0.50
7	0.00	0.00	0.00	0.50	0.50	0.50	1.00	0.58	0.00
8	0.00	0.00	0.00	0.29	0.58	0.29	0.58	1.00	0.58
9	0.00	0.00	0.00	0.00	0.50	0.50	0.00	0.58	1.00

SIMILARITY

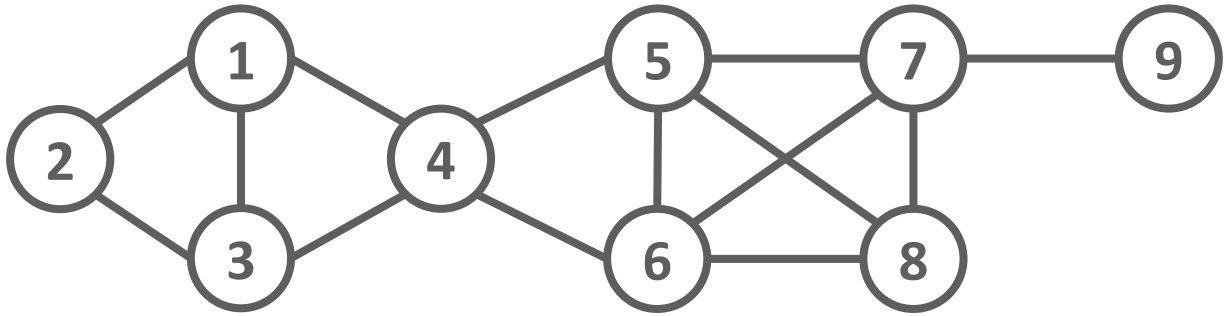
	1	2	3	4	5	6	7	8	9
1	0.00	0.59	0.33	0.71	0.71	0.71	1.00	1.00	1.00
2	0.59	0.00	0.59	0.29	1.00	1.00	1.00	1.00	1.00
3	0.33	0.59	0.00	0.71	0.71	0.71	1.00	1.00	1.00
4	0.71	0.29	0.71	0.00	0.75	0.75	0.50	0.71	1.00
5	0.71	1.00	0.71	0.75	0.00	0.25	0.50	0.42	0.50
6	0.71	1.00	0.71	0.75	0.25	0.00	0.50	0.71	0.50
7	1.00	1.00	1.00	0.50	0.50	0.50	0.00	0.42	1.00
8	1.00	1.00	1.00	0.71	0.42	0.71	0.42	0.00	0.42
9	1.00	1.00	1.00	1.00	0.50	0.50	1.00	0.42	0.00

DISSIMILARITY = 1 - SIMILARITY

- **Step 3**: run the hierarchical clustering algorithm

	1	2	3	4	5	6	7	8	9
1	0.00	0.59	0.33	0.71	0.71	0.71	1.00	1.00	1.00
2	0.59	0.00	0.59	0.29	1.00	1.00	1.00	1.00	1.00
3	0.33	0.59	0.00	0.71	0.71	0.71	1.00	1.00	1.00
4	0.71	0.29	0.71	0.00	0.75	0.75	0.50	0.71	1.00
5	0.71	1.00	0.71	0.75	0.00	0.25	0.50	0.42	0.50
6	0.71	1.00	0.71	0.75	0.25	0.00	0.50	0.71	0.50
7	1.00	1.00	1.00	0.50	0.50	0.50	0.00	0.42	1.00
8	1.00	1.00	1.00	0.71	0.42	0.71	0.42	0.00	0.42
9	1.00	1.00	1.00	1.00	0.50	0.50	1.00	0.42	0.00

DISSIMILARITY

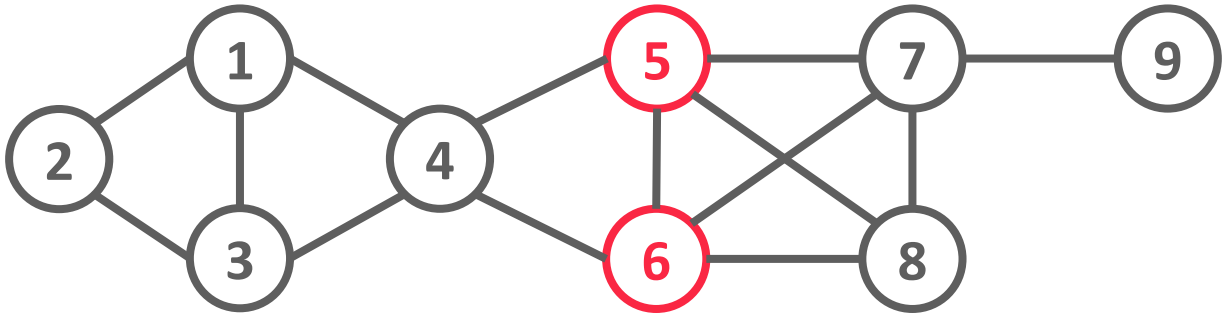


1 2 3 4 5 6 7 8 9

- **Step 3**: run the hierarchical clustering algorithm

	1	2	3	4	5	6	7	8	9
1	0.00	0.59	0.33	0.71	0.71	0.71	1.00	1.00	1.00
2	0.59	0.00	0.59	0.29	1.00	1.00	1.00	1.00	1.00
3	0.33	0.59	0.00	0.71	0.71	0.71	1.00	1.00	1.00
4	0.71	0.29	0.71	0.00	0.75	0.75	0.50	0.71	1.00
5	0.71	1.00	0.71	0.75	0.00	0.25	0.50	0.42	0.50
6	0.71	1.00	0.71	0.75	0.25	0.00	0.50	0.71	0.50
7	1.00	1.00	1.00	0.50	0.50	0.50	0.00	0.42	1.00
8	1.00	1.00	1.00	0.71	0.42	0.71	0.42	0.00	0.42
9	1.00	1.00	1.00	1.00	0.50	0.50	1.00	0.42	0.00

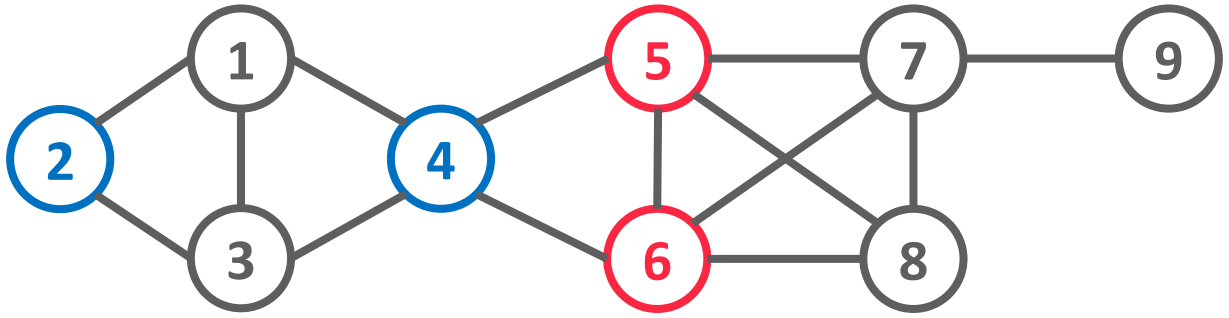
DISSIMILARITY



- **Step 3**: run the hierarchical clustering algorithm

	1	2	3	4	5	6	7	8	9
1	0.00	0.59	0.33	0.71	0.71	1.00	1.00	1.00
2	0.59	0.00	0.59	0.29	1.00	1.00	1.00	1.00
3	0.33	0.59	0.00	0.71	0.71	1.00	1.00	1.00
4	0.71	0.29	0.71	0.00	0.75	0.50	0.71	1.00
5					0.00			
6								
7	1.00	1.00	1.00	0.50	0.50	0.00	0.42	1.00
8	1.00	1.00	1.00	0.71	0.56	0.42	0.00	0.42
9	1.00	1.00	1.00	1.00	0.50	1.00	0.42	0.00

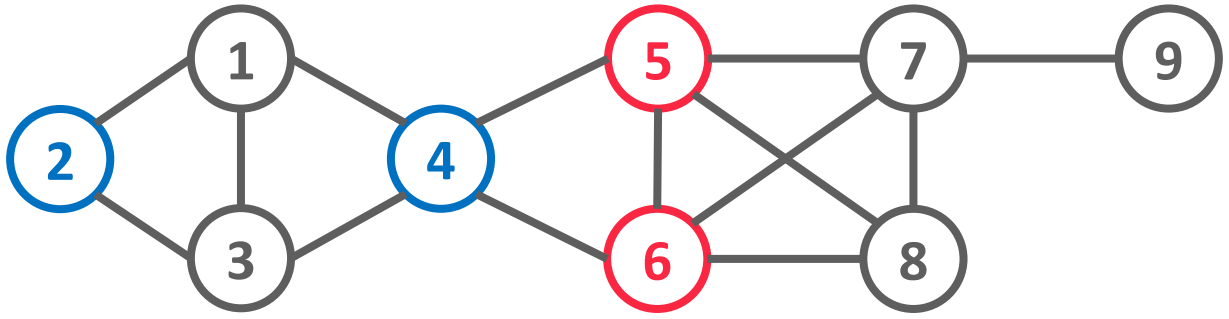
DISSIMILARITY



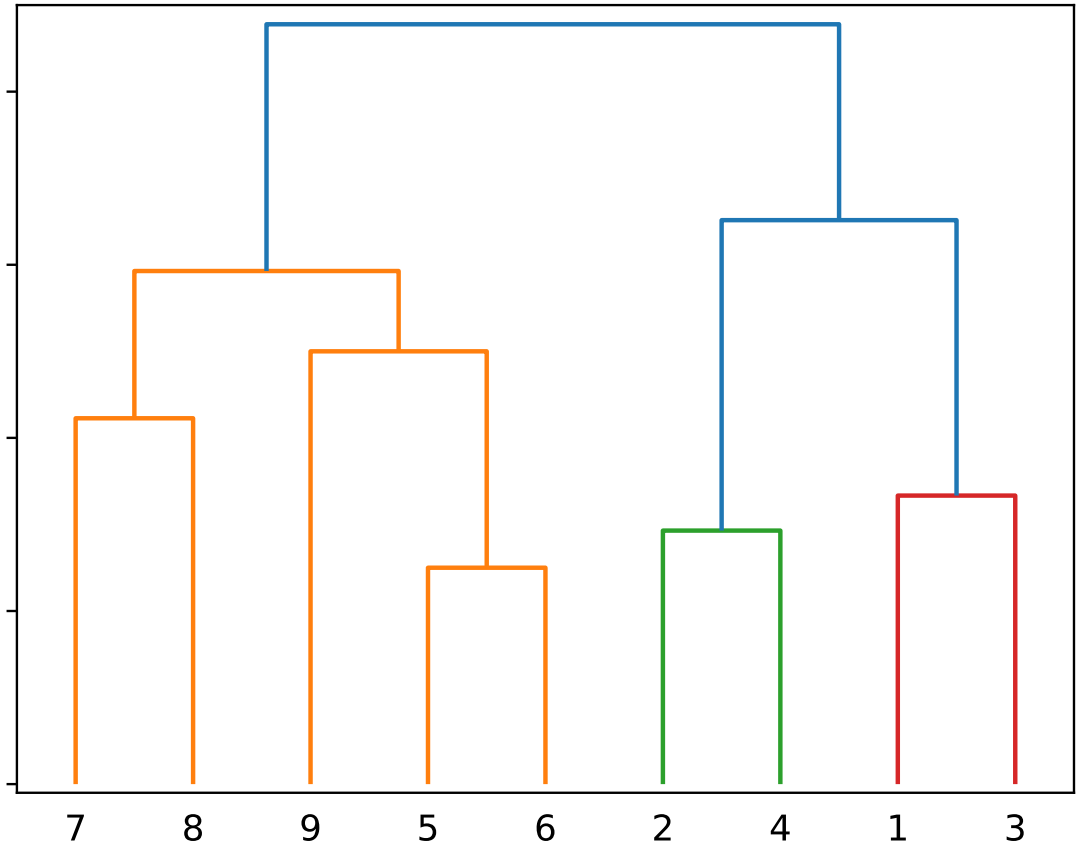
- **Step 3**: run the hierarchical clustering algorithm

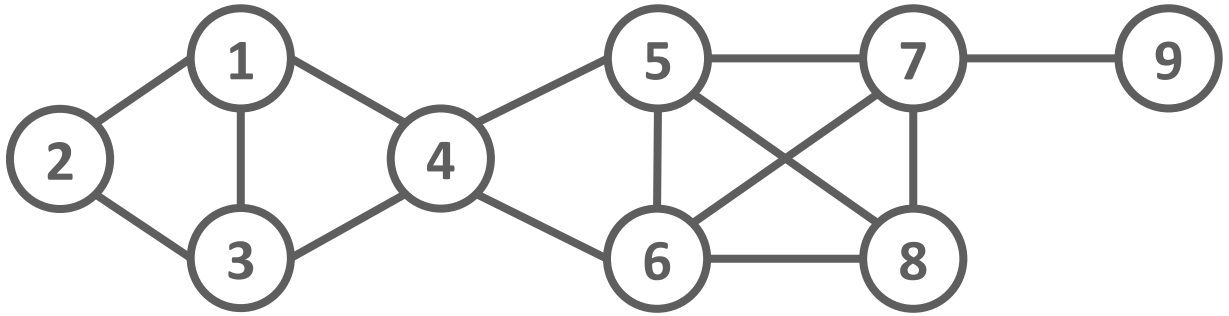
	1	2	3	4	5	6	7	8	9
1	0.00	0.59	0.33	0.71	0.71	1.00	1.00	1.00
2	0.59	0.00	0.59	0.29	1.00	1.00	1.00	1.00
3	0.33	0.59	0.00	0.71	0.71	1.00	1.00	1.00
4	0.71	0.29	0.71	0.00	0.75	0.50	0.71	1.00
5					0.00			
6								
7	1.00	1.00	1.00	0.50	0.50	0.00	0.42	1.00
8	1.00	1.00	1.00	0.71	0.56	0.42	0.00	0.42
9	1.00	1.00	1.00	1.00	0.50	1.00	0.42	0.00

DISSIMILARITY



Hierarchical clustering: agglomerative procedure





Part 2: Community detection

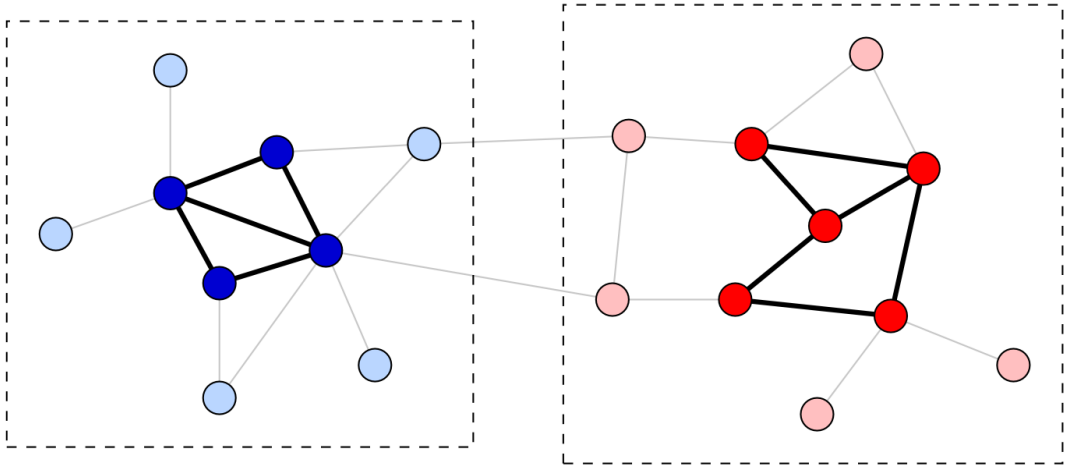
We are going to study four different methods for community detection in graphs.

- Hierarchical clustering
  - Agglomerative **[similarity / dissimilarity]**
  - Divisive **[edge betweenness]**
- Spectral community detection **[modularity]**
- Louvain algorithm **[modularity]**

The agglomerative approach that we have just seen can often have **poor results** in practice.

- One concern is that they **fail** with some frequency to find the correct communities in networks were the community structure is known, which makes it difficult to place much trust in them in other cases.

- They have the tendency to find only the **cores** of communities and leave out the **periphery**. The core nodes in a community often have strong similarity, and hence are connected early in the agglomerative process, but peripheral nodes that have no strong similarity to others tend to get neglected.



Hierarchical clustering: divisive procedure

In [1] the authors propose a different approach to hierarchical clustering of graphs, one which is based on two fundamental ideas:

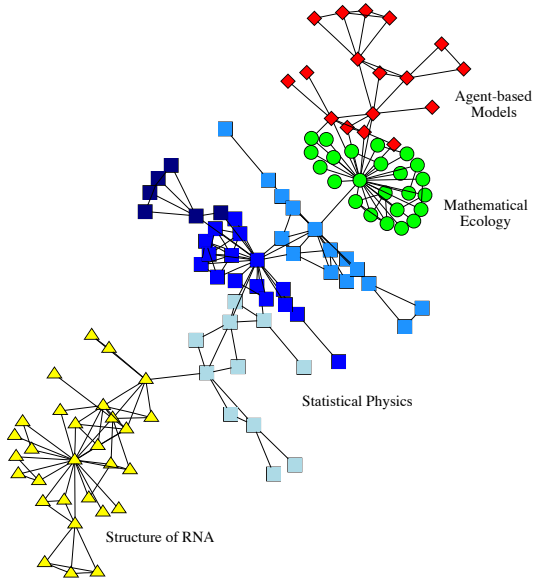
- Run a **divisive** procedure for clustering the vertices of the graph
- Instead of focusing on removing the edges between vertex pairs with lowest similarity (which would be symmetric to the agglomerative approach) we find edges with the highest **betweenness**, where betweenness is some measure that favors edges that lie between communities and disfavors those that lie inside communities.
  - In other words, rather than constructing communities by adding the strongest edges to an initially empty vertex set, we construct them by progressively removing edges from the original graph.

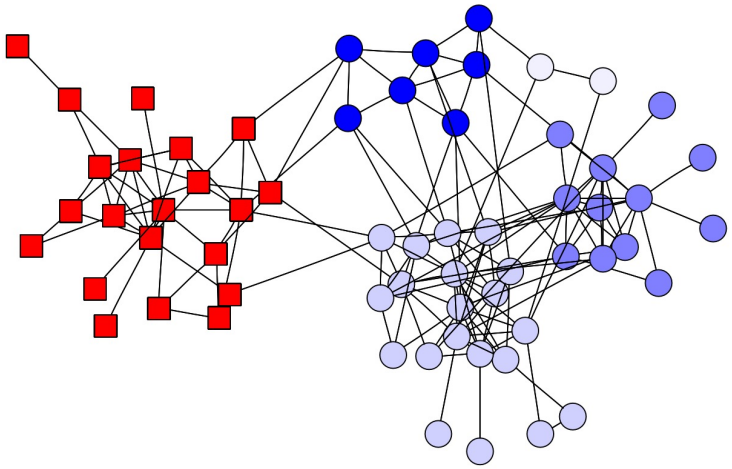
[1] Girvan and Newman "*Community structure in social and biological networks*" (2002)

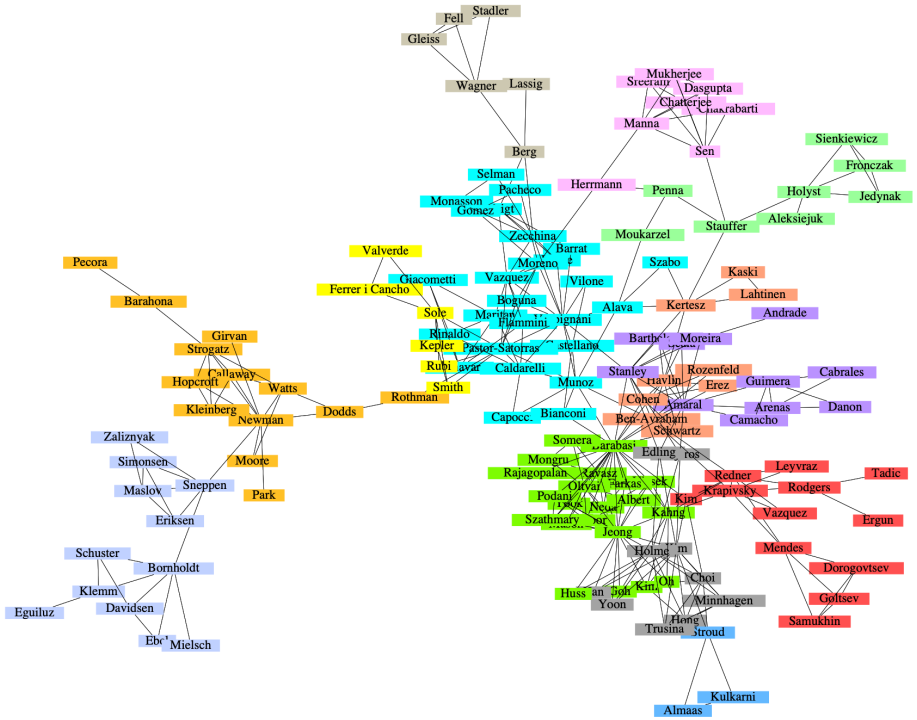
Hierarchical clustering: divisive procedure

Definition: The edge **betweenness** of and edge is the number of shortest paths between pairs of vertices that run along it. If there is more than one shortest path between a pair of vertices, each path is given equal weight such that the total weight of all of the paths is unity.

- If a network contains communities or groups that are only loosely connected by a few intergroup edges, then all shortest paths between different communities must go along one of these few edges. Thus, the edges connecting communities will have high edge betweenness. By removing these edges, we separate groups from one another and so reveal the underlying community structure of the graph.





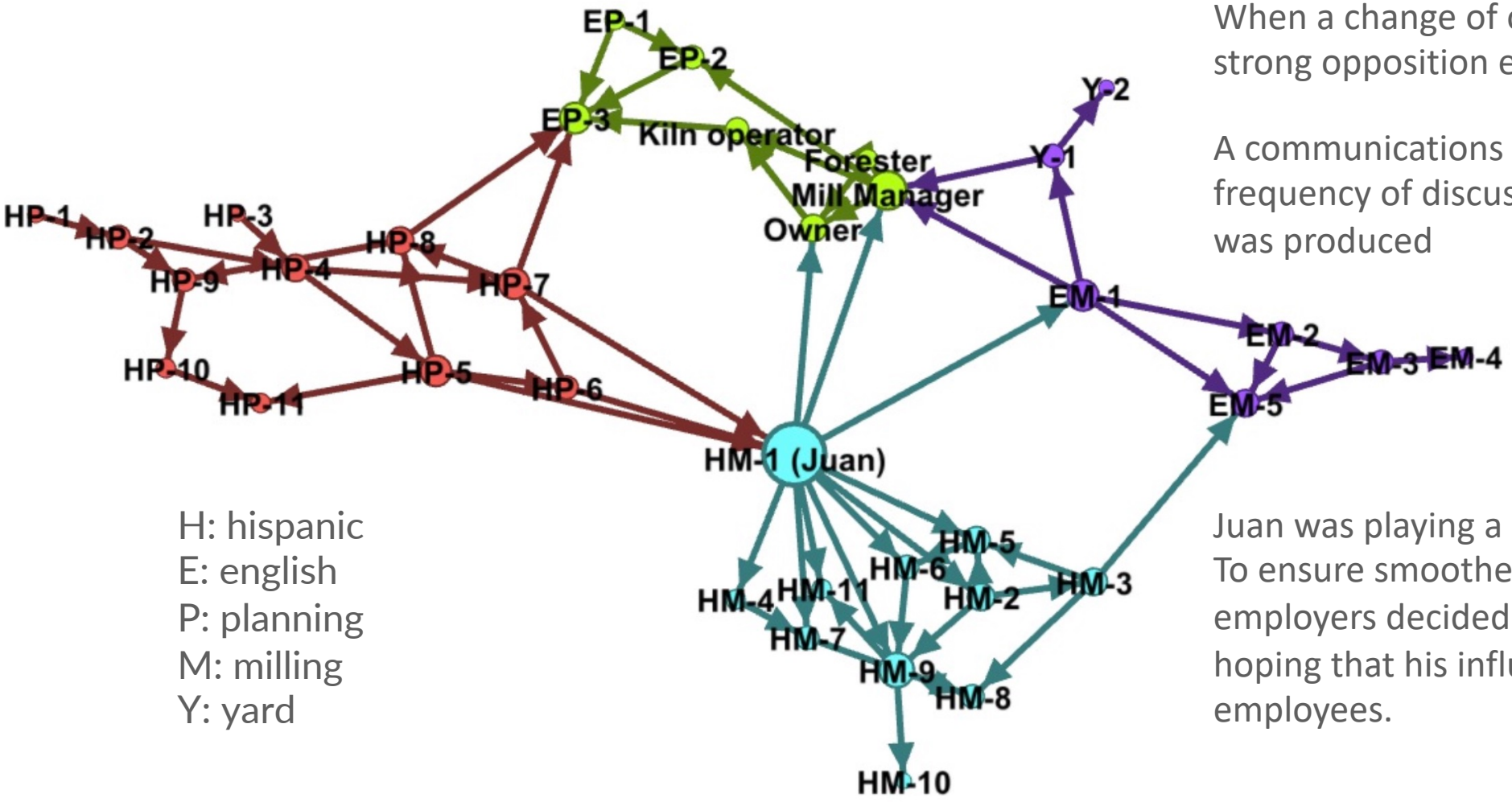


The sawmill network (Michael and Massey, 1997)

Employees were from different ethnic groups (E or H) and assigned to different tasks (P, M, or Y)

When a change of organization was proposed, a strong opposition emerged (i.e. strike)

A communications map, based on the frequency of discussions between employeed was produced



H: hispanic
E: english
P: planning
M: milling
Y: yard

Juan was playing a **central role** in the network. To ensure smoother reactions to changes, the employers decided to persuade Juan first, hoping that his influence would spread to other employees.

The algorithm (Divisive hierarhical clustering with edge betweeness)

1) Calculate betwenness scores for all edges in the network
2) Find the edge with the highest score and remove it from the network
   - If the edge removal splits the graph, then divide the graph into subgroups and compute the edge betweeness of the subgraphs
   - Else update the edge betweeness for the whole graph
3) Repeat Step 2

There are many ways of calculating the betweenness score of an edge, but one that is rather popular is the notion of **beetweenness centrality** defined as follows: the betweeness centrality of edge **e** is the sum of the fraction of all-pairs shortest paths that pass through **e**

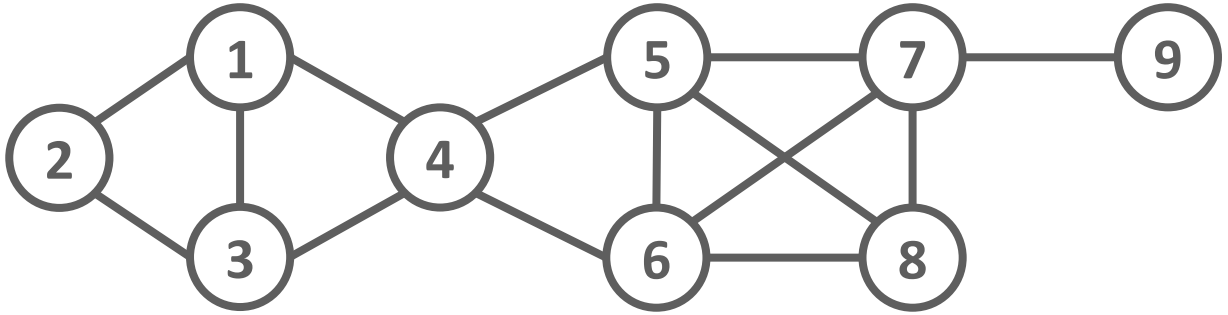
$$c_B(e) = \sum_{s,t \in V} \frac{\sigma(s,t \mid e)}{\sigma(s,t)}$$

number of shortest paths between **s** and **t** passing through edge **e**

number of shortest paths between **s** and **t**

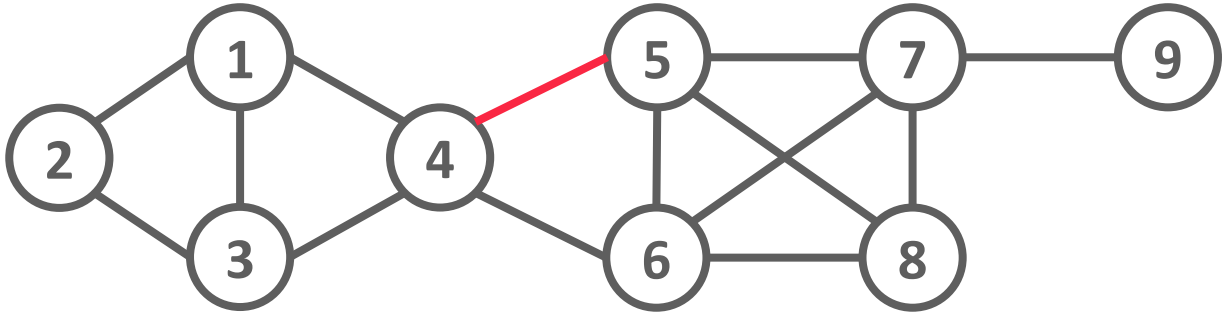
Where **s** and **t** are vertices of the set *V*

Example: in the graph below, the betweeness centrality of each edge is given as



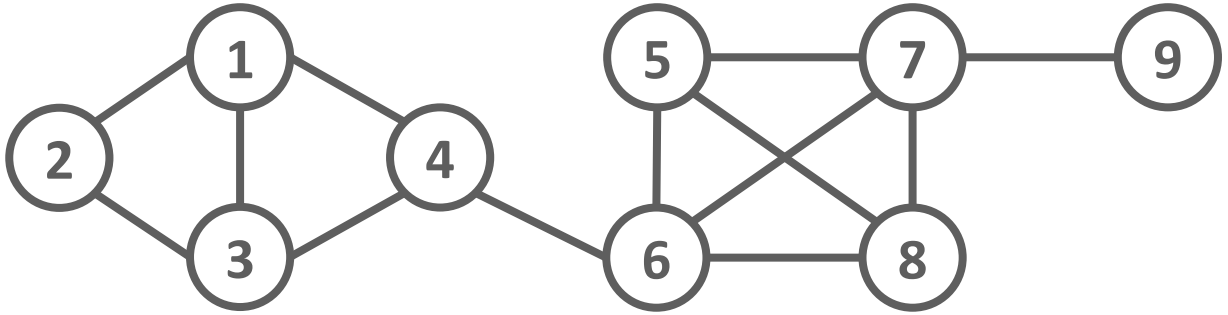
(1, 2) 0.11	(5, 6) 0.03
(1, 3) 0.03	(5, 7) 0.12
(1, 4) 0.25	(5, 8) 0.12
(2, 3) 0.11	(6, 7) 0.12
(3, 4) 0.25	(6, 8) 0.12
(4, 5) 0.28	(7, 8) 0.03
(4, 6) 0.28	(7, 9) 0.11
	(8, 9) 0.11

Example: in the graph below, the betweeness centrality of each edge is given as



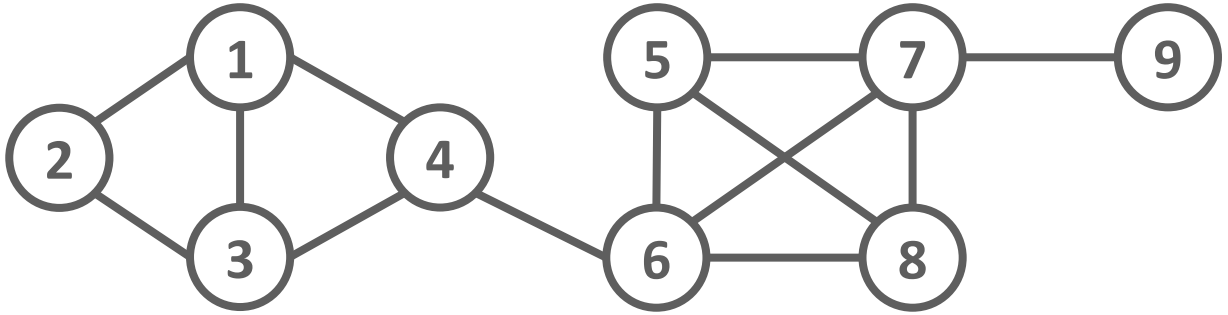
(1, 2) 0.11	(5, 6) 0.03
(1, 3) 0.03	(5, 7) 0.12
(1, 4) 0.25	(5, 8) 0.12
(2, 3) 0.11	(6, 7) 0.12
(3, 4) 0.25	(6, 8) 0.12
(4, 5) 0.28	(7, 8) 0.03
(4, 6) 0.28	(7, 9) 0.11
	(8, 9) 0.11

Example: in the graph below, the betweeness centrality of each edge is given as



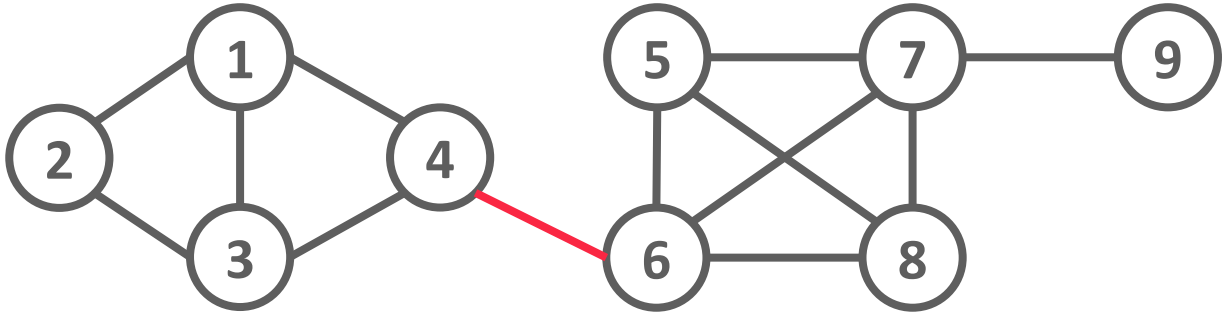
(1, 2) 0.11	(5, 6) 0.03
(1, 3) 0.03	(5, 7) 0.12
(1, 4) 0.25	(5, 8) 0.12
(2, 3) 0.11	(6, 7) 0.12
(3, 4) 0.25	(6, 8) 0.12
(4, 5) 0.28	(7, 8) 0.03
(4, 6) 0.28	(7, 9) 0.11
	(8, 9) 0.11

Example: in the graph below, the betweeness centrality of each edge is given as



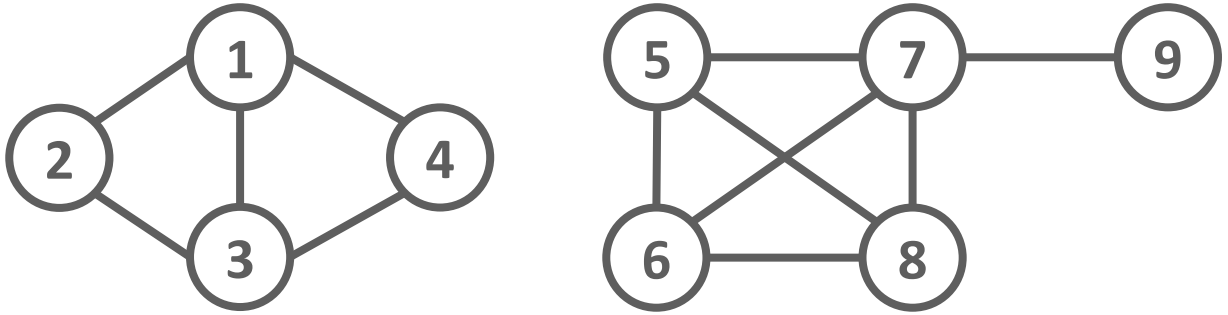
(1, 2) 0.11	(5, 6) 0.14
(1, 3) 0.03	(5, 7) 0.04
(1, 4) 0.25	(5, 8) 0.04
(2, 3) 0.11	(6, 7) 0.21
(3, 4) 0.25	(6, 8) 0.21
(4, 6) 0.56	(7, 8) 0.03
	(7, 9) 0.11
	(8, 9) 0.11

Example: in the graph below, the betweeness centrality of each edge is given as



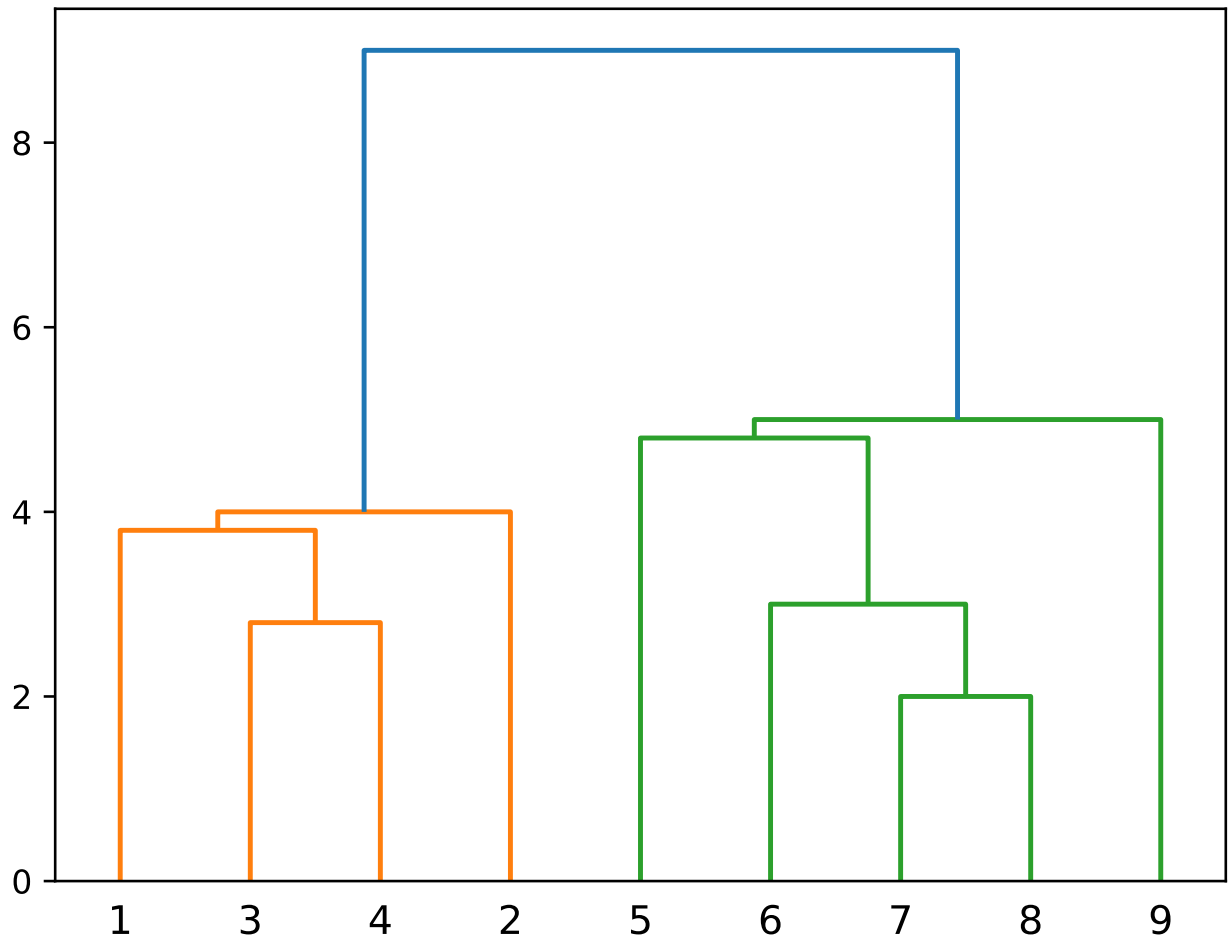
(1, 2) 0.11	(5, 6) 0.14
(1, 3) 0.03	(5, 7) 0.04
(1, 4) 0.25	(5, 8) 0.04
(2, 3) 0.11	(6, 7) 0.21
(3, 4) 0.25	(6, 8) 0.21
(4, 6) 0.56	(7, 8) 0.03
	(7, 9) 0.11
	(8, 9) 0.11

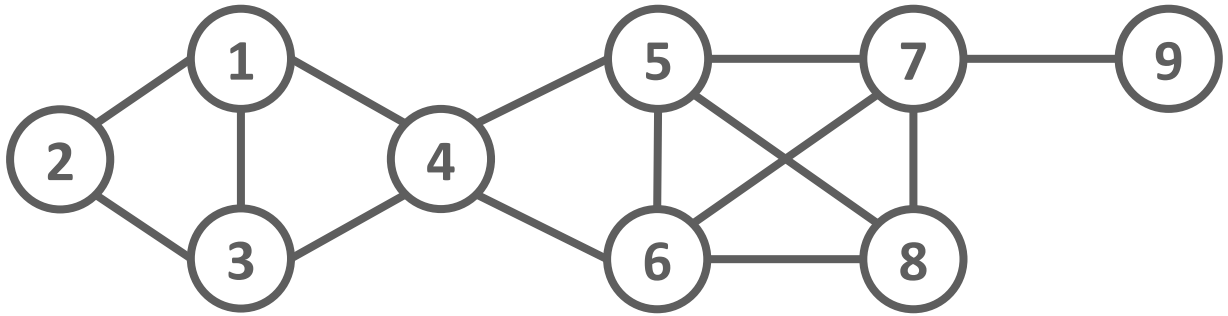
Example: in the graph below, the betweeness centrality of each edge is given as



(1, 2) 0.04	(5, 6) 0.03
(1, 3) 0.03	(5, 7) 0.04
(1, 4) 0.04	(5, 8) 0.04
(2, 3) 0.04	(6, 7) 0.04
(3, 4) 0.04	(6, 8) 0.04
	(7, 8) 0.03
	(7, 9) 0.06
	(8, 9) 0.06

Hierarchical clustering: divisive procedure





Part 2: Community detection

We are going to study four different methods for community detection in graphs.

- Hierarchical clustering
  - Agglomerative **[similarity / dissimilarity]**
  - Divisive **[edge betweenness]**
- Spectral community detection **[modularity]**
- Louvain algorithm **[modularity]**

Part 2: Community detection

We are going to study four different methods for community detection in graphs.

- Hierarchical clustering
  - Agglomerative **[similarity / dissimilarity]**
  - Divisive **[edge betweenness]**
- Spectral community detection **[modularity]**
- Louvain algorithm **[modularity]**

Detecting communities is a hard problem because it is not a very well posed one.

- What does it really mean to aim for communities where there are **many** internal connections and **few** externals ones ?

Detecting communities is a hard problem because it is not a very well posed one.

- What does it really mean to aim for communities where there are **many** internal connections and **few** externals ones ?

The hierarchical clustering methods that we have considered propose communities in a rather intuitive manner.

- But how do we actually **quantify** the quality of the split of their divisions?

Detecting communities is a hard problem because it is not a very well posed one.

- What does it really mean to aim for communities where there are **many** internal connections and **few** externals ones ?

The hierarchical clustering methods that we have considered propose communities in a rather intuitive manner.

- But how do we actually **quantify** the quality of the split of their divisions?

MODULARITY

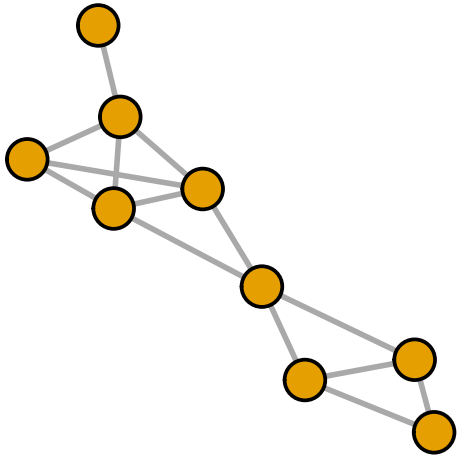
Modularity is the number of edges of a graph falling withing groups minus the expected number in an equivalent network with edges placed at random.

- We find the fraction of edges that run between vertices of the same type, and then we subtract from that number the fraction of such edges we would *expect* to find if edges were positioned at random without regard for vertex type

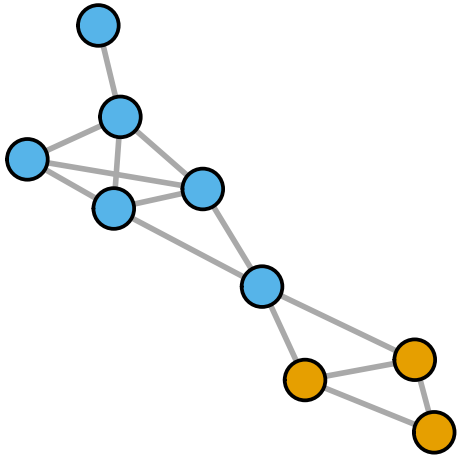
Modularity is a measure of the **extent to which like is connected to like in a network**. It is strictly less than 1 and takes positive values if there are more edges between nodes of the same type than we would expect by random chance. It can also take negative values if there are fewer such edges than we would expect by chance.

Spectral community detection

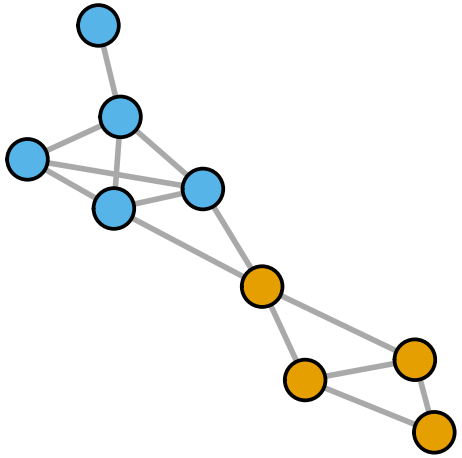
Q = 0



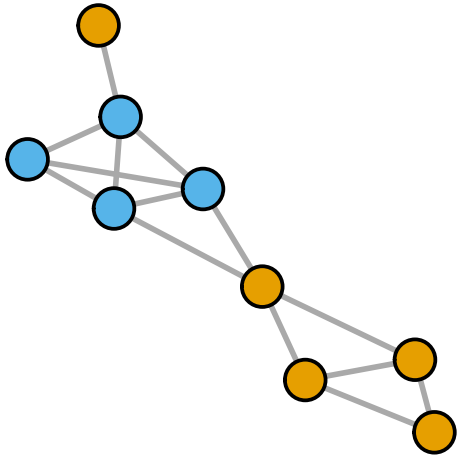
Q = 0.265



Q = 0.347



Q = 0.283



Consider a graph **G** with ***n*** vertices and ***m*** edges and adjacency matrix **A**

Suppose we split the graph in ***N*** groups so vertex ***i*** belongs to group $g_i \in \{1, \ldots, N\}$

The total number of edges that run between nodes of the same group is

$$\sum_{\text{edges}(i,j)} \delta_{g_i g_j} = \frac{1}{2} \sum_{i,j} A_{ij} \; \delta_{g_i g_j}$$

We count edges in double in this summation

Zero if different groups, one if the same

Consider now a vertex ***i*** with degree k_i

For an edge e_i that starts at vertex ***i*** there are **2m-1** positions where it might end

If we consider that edges are created **randomly**, then we can say that the probability that e_i ends at vertex ***j*** is $k_j/2m$ and that the expected number of edges between vertex ***i*** and ***j*** will be $k_i k_j/2m$

The expected total number of edges between vertices of the same type in a random graph is therefore

$$\frac{1}{2}\sum_{i,j}\frac{k_i k_j}{2m}\ \delta_{g_i g_j}$$

We usually prefer talk in terms of fractions, so we end up dividing everything by ***m***

The modularity of the network is therefore given as

$Q =$ **Fraction of edges between vertices of the same group in the graph under analysis** $-$ **Expected fraction of edges between vertices of the same group in a graph with edges chosen randomly**

We usually prefer talk in terms of fractions, so we end up dividing everything by ***m***

The modularity of the network is therefore given as

$$Q = \frac{1}{m}\left(\frac{1}{2}\sum_{i,j} A_{ij}\ \delta_{g_i g_j}\right) - \frac{1}{m}\left(\frac{1}{2}\sum_{i,j} \frac{k_i k_j}{2m}\ \delta_{g_i g_j}\right)$$

We usually prefer talk in terms of fractions, so we end up dividing everything by ***m***

The modularity of the network is therefore given as

$$Q = \frac{1}{m}\left(\frac{1}{2}\sum_{i,j} A_{ij}\ \delta_{g_i g_j}\right) - \frac{1}{m}\left(\frac{1}{2}\sum_{i,j} \frac{k_i k_j}{2m}\ \delta_{g_i g_j}\right)$$

$$Q = \frac{1}{2m}\sum_{i,j}\left(A_{ij} - \frac{k_i k_j}{2m}\right)\ \delta_{g_i g_j}$$

Brute force: now that we have a quantitative way of assessing the quality of the split of a graph into communities, we could envision an algorithm that would test **all** possible configurations of a graph and then return the one with the highest modularity.

- The **problem** with such approach is that if we have n vertices and want to split them into two classes, we need to test 2^n different combinations. For a not-so-big graph with 100 vertices, this would mean calculating the modularity over 10^{30} different graphs!

All practical algorithms propose **heuristic** ways of approximately maximizing the modularity. We will see two examples of such methods now.

Part 2: Community detection

We are going to study four different methods for community detection in graphs.

- Hierarchical clustering
  - Agglomerative **[similarity / dissimilarity]**
  - Divisive **[edge betweenness]**
- Spectral community detection **[modularity]**
- Louvain algorithm **[modularity]**

Consider that we want to split the graph just in two communities.

We construct a vector ***s*** of ***n*** dimensions (number of vertices) and each of its elements is either +1 or -1, indicating to which community the vertex belongs to. Noticing that we can write in this case

$$\delta_{g_i g_j} = \frac{1}{2}(s_i s_j + 1)$$

and therefore

$$Q = \frac{1}{2m} \sum_{i,j} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \frac{1}{2}(s_i s_j + 1)$$

Consider that we want to split the graph just in two communities.

We construct a vector ***s*** of ***n*** dimensions (number of vertices) and each of its elements is either +1 or -1, indicating to which community the vertex belongs to. Noticing that we can write in this case

$$\delta_{g_i g_j} = \frac{1}{2}(s_i s_j + 1)$$

and therefore

$$Q = \frac{1}{4m} \sum_{i,j} \left(A_{ij} - \frac{k_i k_j}{2m} \right) (s_i s_j + 1)$$

Consider that we want to split the graph just in two communities.

We construct a vector ***s*** of ***n*** dimensions (number of vertices) and each of its elements is either +1 or -1, indicating to which community the vertex belongs to. Noticing that we can write in this case

$$\delta_{g_i g_j} = \frac{1}{2}(s_i s_j + 1)$$

and therefore

$$Q = \frac{1}{4m} \sum_{i,j} \left(A_{ij} - \frac{k_i k_j}{2m} \right) s_i s_j$$

Consider that we want to split the graph just in two communities.

We construct a vector ***s*** of ***n*** dimensions (number of vertices) and each of its elements is either +1 or -1, indicating to which community the vertex belongs to. Noticing that we can write in this case

$$\delta_{g_i g_j} = \frac{1}{2}(s_i s_j + 1)$$

and therefore

$$Q = \frac{1}{4m}\boldsymbol{s}^T \boldsymbol{B}\boldsymbol{s} \quad \text{with} \quad B_{ij} = A_{ij} - \frac{k_i k_j}{2m}$$

Our goal is to maximize the modularity Q, so we can see our problem as that of finding the values of the vector ***s*** for a given matrix ***B*** encoding the structure of the graph.

- One reason why this maximization problem is a **difficult** one is that the s_i can take only a discrete set of values—each s_i is restricted to the values +1 and -1. If, on the other hand, the s_i were continuous-valued variables, allowed to take any real values, the problem would be much easier—we could just **differentiate** to the find the maximum.

We follow a **relaxation** approach: we consider that the s_i can have values in the real line and assume that their values will give as us good clues as to how to split the graph into -1 and +1 classes.

- If ***s*** is to be seen as a n-dimensional vector, the requirement of its values being always either +1 or -1 means that such vector always points to one of the corners of an n-dimensional hypercube. When we relax the discrete-value constraint, we should impose that the sum of the squares of the components of ***s*** is n, i.e.

$$\sum_i s_i^2 = n$$

So now community detection problem becomes an optimization problem under constraints

$$\underset{\text{s.t.}\|\boldsymbol{s}\|^2=n}{\text{maximize}} \quad \frac{1}{4m}\boldsymbol{s}^T\boldsymbol{B}\boldsymbol{s}$$

with an augmented Lagrangian given by

$$\mathcal{L}(\boldsymbol{s},\lambda) = \frac{1}{4m}\boldsymbol{s}^T\boldsymbol{B}\boldsymbol{s} + \lambda(n - \boldsymbol{s}^T\boldsymbol{s})$$

Taking the derivatives we see that the vector that maximizes the loss function is…

Taking the derivatives we see that the vector that maximizes the loss function is…

$$\boldsymbol{B}\boldsymbol{s} = \beta \boldsymbol{s}$$

In other words, the optimal ***s*** is one of the eigenvectors of the modularity matrix **B** and β is the corresponding eigenvalue.

- To know which eigenvalue to use, we can substitute the equation above into the definition of the modularity and see that we should take the **largest** eigenvalues.

$$Q = \frac{1}{4m}\boldsymbol{s}^T\boldsymbol{B}\boldsymbol{s} = \frac{1}{4m}\beta\boldsymbol{s}^T\boldsymbol{s} = \frac{n}{4m}\beta$$

Question: How to relate this vector of real values to the actual **discrete-valued** vector?

Question: How to relate this vector of real values to the actual **discrete-valued** vector?

Take the signs of the elements of $\boldsymbol{s}$ and assign the communities accordingly !

- **Attention**: this will not necessarily give the global solution to the modularity maximization problem, but it is a good heuristic. It is interesting to note that the **intensities** of the elements of the vector have an interpretation: these values indicate how firmly we assign each vertex to one community or the other. Those with large values are strong members of the communities, whereas smaller values are more ambivalent members.

Question: How to relate this vector of real values to the actual **discrete-valued** vector?

Take the signs of the elements of ***s*** and assign the communities accordingly !

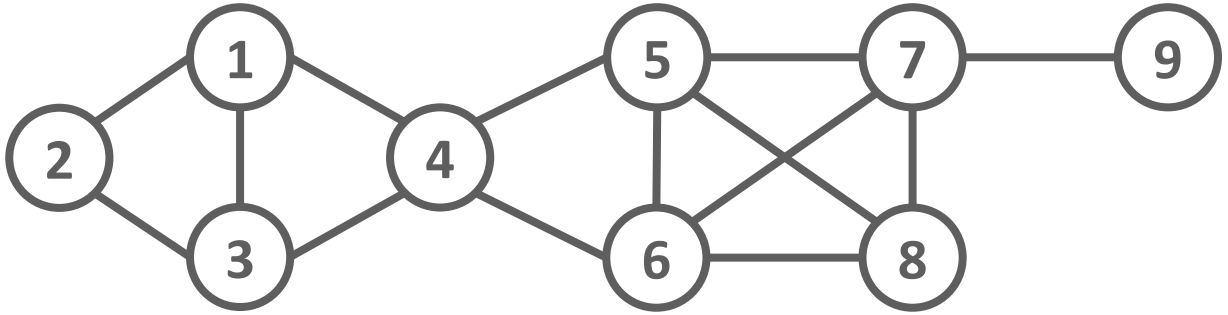
- **Attention**: this will not necessarily give the global solution to the modularity maximization problem, but it is a good heuristic. It is interesting to note that the **intensities** of the elements of the vector have an interpretation: these values indicate how firmly we assign each vertex to one community or the other. Those with large values are strong members of the communities, whereas smaller values are more ambivalent members.

The algorithm (Spectral community detection)

1) Build the modularity matrix **B**
2) Calculate the eigenvector of this matrix corresponding to the largest eigenvalue
3) Assign vertices to communities according to the signs of their elements, i.e. positive signs in one group and negative signs in the other group

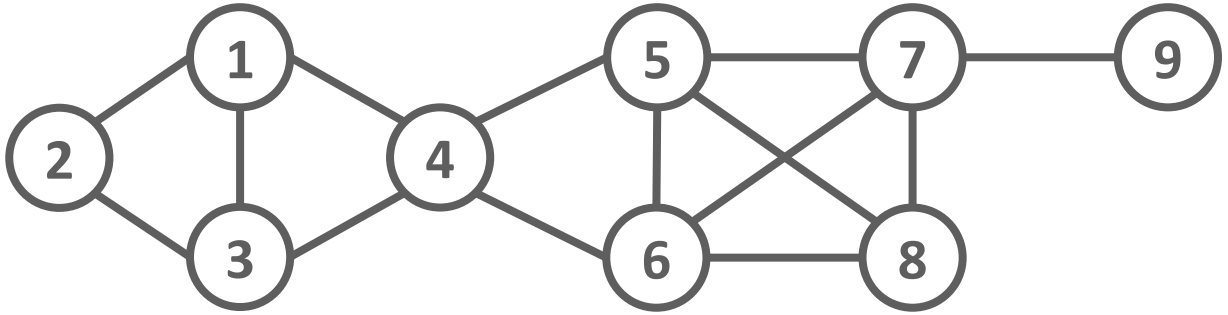
Newman (2013) – "Spectral methods for network community detection and graph partitioning"

Example: in the graph below, the leading eigenvector is



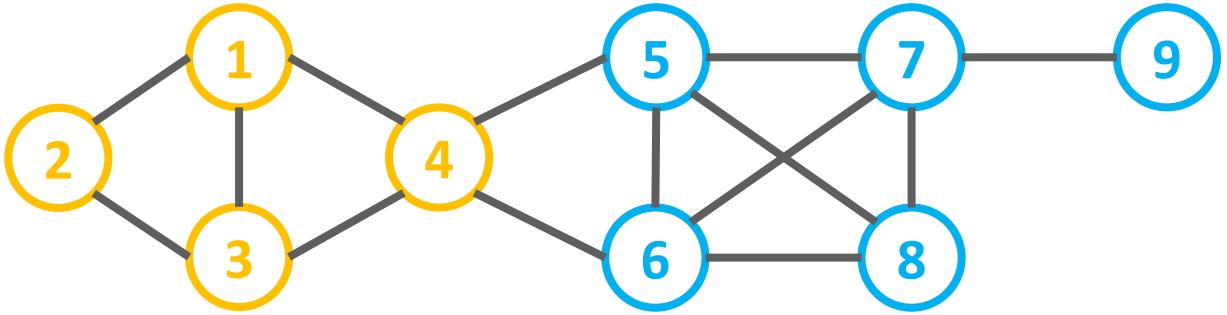
+0.4382574
+0.3792501
+0.4382574
+0.1738957
-0.2853926
-0.2853926
-0.3754298
-0.3444000
-0.1390457

Example: in the graph below, the leading eigenvector is



+0.4382574	+1
+0.3792501	+1
+0.4382574	+1
+0.1738957	+1
-0.2853926	-1
-0.2853926	-1
-0.3754298	-1
-0.3444000	-1
-0.1390457	-1

Example: in the graph below, the leading eigenvector is



	+0.4382574	+1
	+0.3792501	+1
	+0.4382574	+1
Ambivalent vertex →	+0.1738957	+1
	-0.2853926	-1
	-0.2853926	-1
	-0.3754298	-1
	-0.3444000	-1
	-0.1390457	-1

It is interesting to note that the modularity matrix **B** defined as follows

$$B_{ij} = A_{ij} - \frac{k_i k_j}{2m}$$

has two important properties:

$$\sum_i B_{ij} = \sum_i A_{ij} - \frac{k_j}{2m} \sum_i k_i = k_j - \frac{k_j}{2m} 2m = 0$$

$$\sum_j B_{ij} = \sum_j A_{ij} - \frac{k_i}{2m} \sum_j k_j = k_i - \frac{k_i}{2m} 2m = 0$$

Which means that such matrix **always** have a zero eigenvalue associated to vector $\mathbf{s} = \mathbf{1}_N$

It is interesting to note that the modularity matrix **B** defined as follows

$$B_{ij} = A_{ij} - \frac{k_i k_j}{2m}$$

Remarks:

- Matrix **B** always have a zero eigenvalue associated to vector $\mathbf{s} = \mathbf{1}_N$
- Matrix **B** is symmetric but not necessarily positive definite. Therefore, it always has real eigenvalues (which can be positive or negative) and real eigenvectors.

Conclusion:

If the largest eigenvalue of **B** is zero, it means that there is no way of splitting the graph into communities with a higher modularity than a random graph.

The spectral approach to community detection is a **relaxation heuristic** to the solution of the modularity maximization problem. Therefore, the results are not necessarily the global optimum (although it works really well in practice!). It is possible to **fine-tune** the output of the algorithm using a second simple algorithm for community detection:

The algorithm (Fine-tuning of the spectral algorithm)

1) Split the graph using the leading eigenvector of **B**
2) For each node in the graph, calculate how much the modularity would change if it was moved from one group to another group.
3) Choose among these changes the node for which the modularity increases the most. Change the group of that node.
4) Repeat **Steps 2** and **3** without moving the nodes that have already been moved.
5) When all nodes have been moved exactly once, finish the current round.
6) Go through all of the changes in positions in the round that has just ended and choose the configuration with the largest modularity. Go back to **Step 1** using this split of the graph as starting point for another round (i.e. **Steps 2-6**)
7) Stop the algorithm when the modularity no longer improves. Use the division with the largest modularity of the last round.

Remarks:

- For a graph with **n** vertices and **m** edges, the node-moving strategy has time complexity of **$O(n(m+n))$** or simply **$O(n^2)$** for sparse matrices (i.e. with few edges)

- If done naively, the time complexity of the spectral approach would be the same as that for the node-moving approach. However, the matrix operations of this procedure can be simplified by taking into account that a vertex usually has only a few neighbors. The actual complexity of the algorithm is actually **$O(n \log(n))$** for sparse graphs.

- So far we have only considered graphs being split into two communities. Extending the spectral approach to more communities can be done via **bissection**: we first find two main communities and then re-run the algorithm on each of these communities to find two other communities, etc.

- The node-moving approach extends more naturally to more groups, but is rather slow.

Part 2: Community detection

We are going to study four different methods for community detection in graphs.

- Hierarchical clustering
  - Agglomerative **[similarity / dissimilarity]**
  - Divisive **[edge betweenness]**
- Spectral community detection **[modularity]**
- Louvain algorithm **[modularity]**

The Louvain algorithm

The **Louvain** algorithm is inspired by the node-moving approach as presented before, but manages to have a time complexity equivalent to that of the spectral approach.

In other words, it has the advantage of extending naturally to multiple groups (as in the node-moving procedure) while having a time complexity equivalent to that of the spectral approach, i.e. **O(n log(n))**

Because of its numerical advantages, the Louvain method is one of the most important and used algorithms for community detection nowadays. In the paper where they proposed the method, the authors of the algorithm claimed that "Contrary to all the other community detection algorithms, the network size limits that we are facing with our algorithm are due to **limited storage capacity rather than limited computation time**: identifying communities in a 118 million nodes network took only 152 minutes."

Blondel et al. (2008) – "Fast unfolding of communities in large networks"

Assume we have a graph with ***n*** nodes.

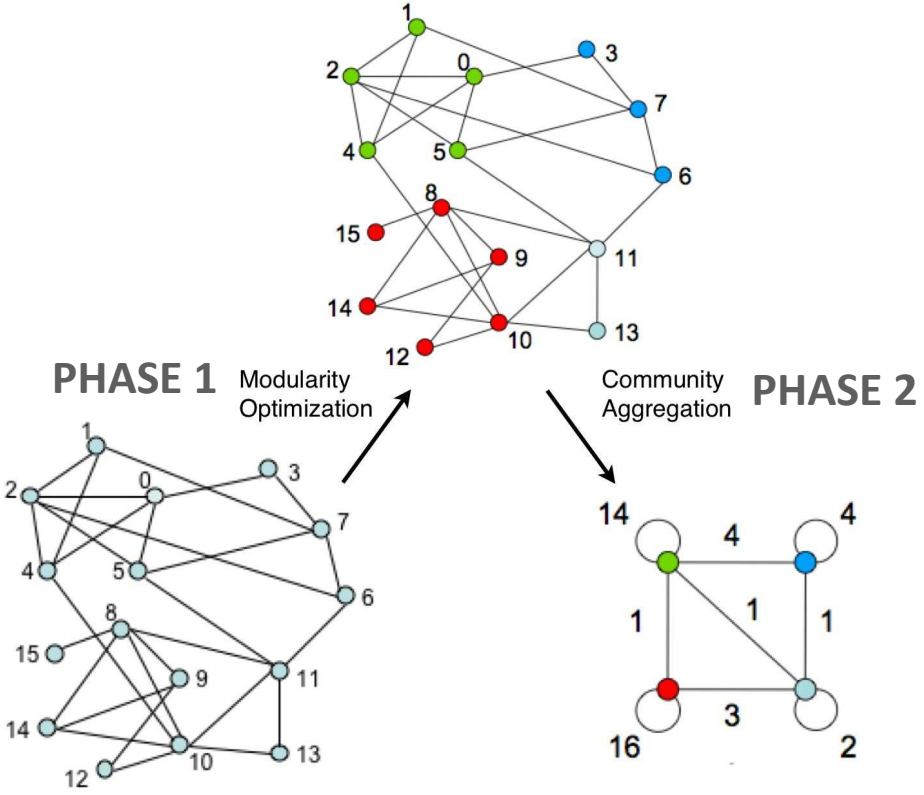
The **Louvain** algorithm consists of **two phases** that are repeated iteratively:

Phase 1: First, we assign a different community to each node of the network. In this initial partition there are as many communities as there are nodes.

Then, for each node ***i*** we consider the neighbours ***j*** of ***i*** and we evaluate the gain of modularity that would take place by removing ***i*** from its community and by placing it in the community of ***j***. The node ***i*** is then placed in the community for which this gain is maximum (in case of a tie we use a breaking rule), but only if this gain is positive. If no positive gain is possible, ***i*** stays in its original community.

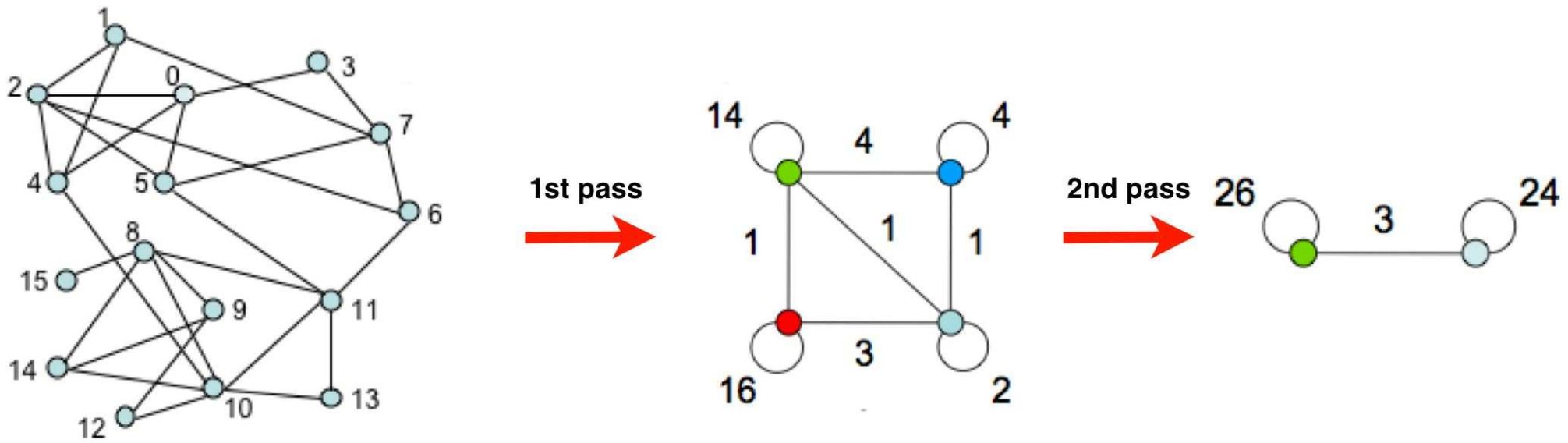
This process is applied repeatedly and sequentially for all nodes until no further improvement can be achieved. The first phase stops when a local maxima of the modularity is attained, i.e., when no individual move can improve the modularity.

Phase 2: The second phase of the algorithm consists in building a new network whose nodes are now the communities found during the first phase. To do so, the weights of the links between the new nodes are given by the sum of the weight of the links between nodes in the corresponding two communities. Links between nodes of the same community lead to self-loops for this community in the new network.



Once the second phase is **completed**, it is possible to reapply the first phase of the algorithm to the resulting weighted network and to iterate.

If we denote “pass” a combination of these two phases, we see that by construction the number of meta-communities decreases at each pass, and as a consequence most of the computing time is used in the first pass. The passes are iterated until there are no more changes and a maximum of modularity is attained.



The Louvain algorithm has several **advantages**:

1) Its steps are intuitive and easy to implement, and the outcome is unsupervised.

2) The algorithm is extremely fast, i.e., computer simulations on large ad-hoc modular networks suggest that its complexity is linear on typical and sparse data.

3) The algorithm provides a decomposition of the network into communities for different levels of organization. This means that the intermediate solutions found by the algorithm may also be meaningful and that the uncovered hierarchical structure may allow the end-user to zoom in the network to observe its structure with the desired resolution.

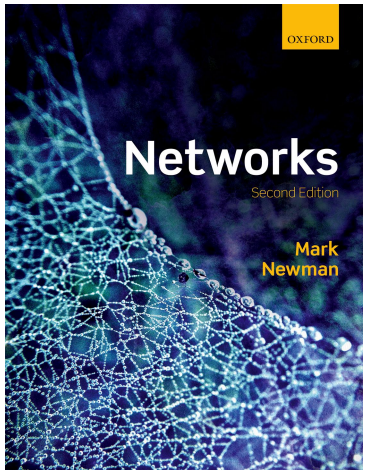
Part 2: Community detection

We are going to study four different methods for community detection in graphs.

- Hierarchical clustering
  - Agglomerative **[similarity / dissimilarity]**
  - Divisive **[edge betweenness]**
- Spectral community detection **[modularity]**
- Louvain algorithm **[modularity]**

CONCLUSION

- We have seen some basic concepts around graphs and their properties
- We have seen algorithms for community detection in graphs
  - Important concept: **modularity**
- There are many other important concepts around graphs, such as:
  - Graph embedding
  - Vertex subsampling
  - Percolation and network resilience, etc.



Mark Newman "Networks: an introduction"
See Chapter 14 for this course