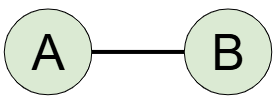
**Louvain + Leiden clustering**

This is a *graph*-based algorithm which groups **nodes** (patients) into *tightly-connected clusters* called **communities**. It quantifies how well the nodes are clustered into communities with the aid of **modularity**.

**Step 1 Determine a correlation between each pair of nodes (patients)**

In Louvain & Leiden clustering, a correlation between 2 nodes is denoted by an **edge**. 

These edges are ***UNdirectional*** (do not have any direction) and ***UNweighted*** (no weight is assigned to the edge, meaning it will not quantify the *strength* of the correlation).

The most common way that the algorithm decides whether an edge should exist between 2 patients (represented as **nodes**) is via the **Pearson Correlation coefficient** (**r**). You may set thresholds for **r** i.e. if **r** < 0.95 between nodes A and B, then do NOT let there be an edge.

**Step 2 Modularity optimisation: Single-node movements → Applies to Leiden + Louvain**

You now have a set of interconnected nodes, with each node representing a unique community.

The Louvain & Leiden algorithms will both iterate over **each node** to check if moving that node to any other community increases **Q**. If moving the node does increase Q, then the algorithm does this.

This process **stops** when no more single-node moves can increase **Q**.

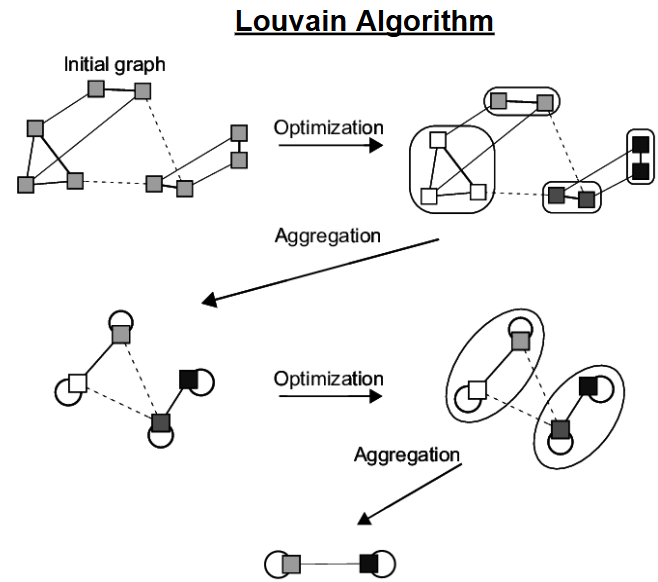
**Step 3 - Partition refinement → Applies ONLY to Leiden clustering**

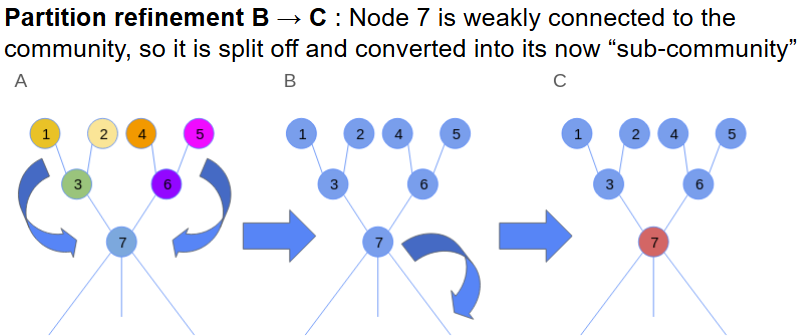
A key issue identified for the Louvain algorithm was that communities may be “internally disconnected” but still remained as one community. So, the Leiden algorithm further partitions the nodes within each community to split them into “sub-communities” (even if that means increasing Q). This is vital, as it ensures all nodes within each community (and sub-community) are very tightly connected to each other.

**Step 4 Community aggregation → Applies to Leiden + Louvain**

After each node has been designated to a certain community / sub-community, the nodes in each group are *aggregated* into 1 single node.

Each node (representing each community) is then subject to further *optimization* , *partition refinement* and *aggregation*, until you end up with a final network with a maximised **Q** value.



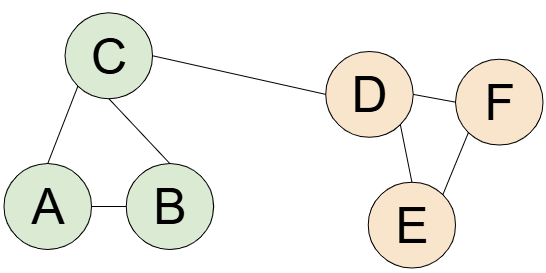


**How is modularity calculated for an undirectional & unweighted network?**

The Standard Modularity (Q) is given by this formula (in both, Leiden and Louvain):

]

* **Q**: Modularity
* **m**: Number of edges in the network
* **2m**: The summation of the degree of each node
* **Aij**: The **adjacency** matrix, where Aij = 1 if an edge exists between **i** and **j**, or Aij = 0 if otherwise
* **Ki**: The number of edges that node **i** is a part of (**degree** of i)
* **Ci**: The community that the node **i** belongs to
* **δ**(**Ci , Cj**): If node **i** & **j** belong to the same community, then this outputs 1 (allowing the sum of that expression to not be 0, but if they belong to different communities, then the sum is 0)
* (**ki** ✕ **kj**) **/ 2m**: The *expected* number of edges between **i** & **j** (this value is between 0 and 1) in a random network where each node retains the same **degree** as in the original Louvain cluster.

For example, if we run a Louvain algorithm and obtain the following graph.

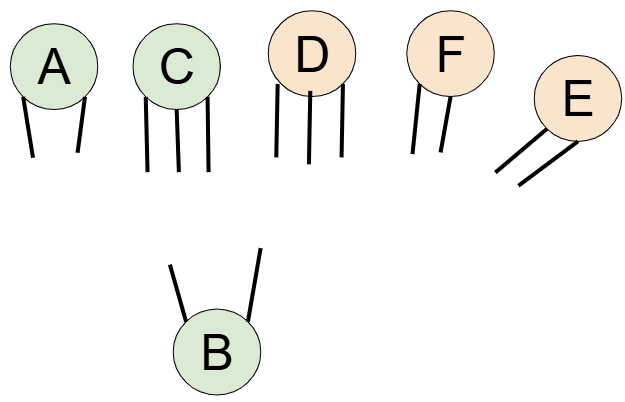
There are 6 nodes (A - F). So, let us write out the **Adjacency** matrix.

There are 2 communities **Green** and **Orange**.

**m** = 7 , **2m** = 14

Let’s look at ABC. There exists a connection between **B** and **C**.

| **Adj** | A | B | C | D | E | F |
| --- | --- | --- | --- | --- | --- | --- |
| A | **0** | 1 | 1 | 0 | 0 | 0 |
| B | 1 | **0** | 1 | 0 | 0 | 0 |
| C | 1 | 1 | **0** | 1 | 0 | 0 |
| D | 0 | 0 | 1 | **0** | 1 | 1 |
| E | 0 | 0 | 0 | 1 | **0** | 1 |
| F | 0 | 0 | 0 | 1 | 1 | **0** |



In a random network, we assume that the degree of each node is constant. So, now, we ask, what are the chances of **B** being paired with **C** here?

We can calculate this via the **configuration model**, and get 6 / 14 = **0.428…**

So, this is our expected no. of edges (in a way). That means, the Louvain algorithm creates **0.572** edges more than what is expected by chance, between **B** and **C**. This difference is added to a *running total*.

This is done for all pairs of nodes belonging to the same community

(A + B, B + C, A + C, D + E, E + F, D + F) but **not** for C + D (as they are of different communities).

Thus, the *running total* keeps track of:

“ how much more / less **intra**-community connectivity exists compared to a random network “

This running total is then **normalised** (multiplied by **1/2m**) to yield a value between **-0.5 < Q < 1**.

**Weighted network**

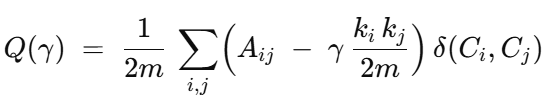
If the edges are **weighted**, **Q** is calculated similarly, except:

* **Aij** contains entries of the *weights* of each edge between **i** and **j** (instead of a binary 0 or 1).
* **Ki**: The sum of the weights of each edge that node **i** is a part of (**degree** of i)
* **m**: Sum of all edge weights

**Resolution limit**

There is another term in the formula for **Q** which we did not discuss: the **resolution parameter γ**.

This parameter is only included within the **Leiden** algorithm , and it controls community granularity !!!

**γ scales** the Expected no. of edges ↑ or ↓. We normally take it to be 1, but it can be raised or lowered.

* **Raising γ (>1)**: This makes it harder for 2 nodes in the same community to have an edge weight above expectation, so we end up with a **larger number** of **small communities**.
* **Lowering γ (<1):** Makes it easier for 2 nodes in the same community to have an edge weight above expectation, so we end up with a **smaller number** of **large communities**.

**Tunable parameters in Louvain / Leiden clustering**

* Altering the resolution parameter (only applicable to Leiden clustering)
* Altering the criteria for an edge to form
* Letting edges be weighted or unweighted