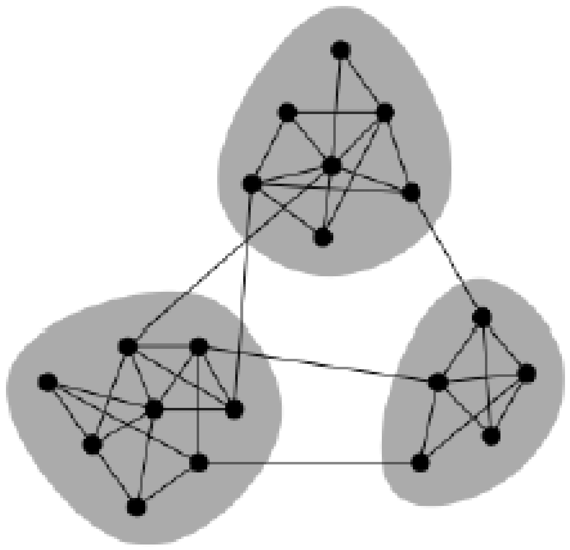
Community detection

Network communities

Definition

Network communities are groups of vertices such that vertices inside the group connected with many more edges than between groups.

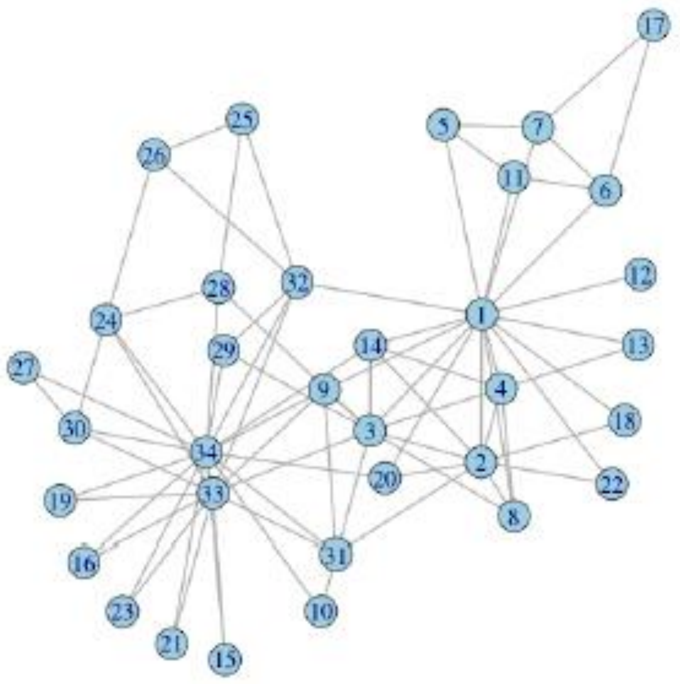


- Graph partitioning problem

Network communities

Definition

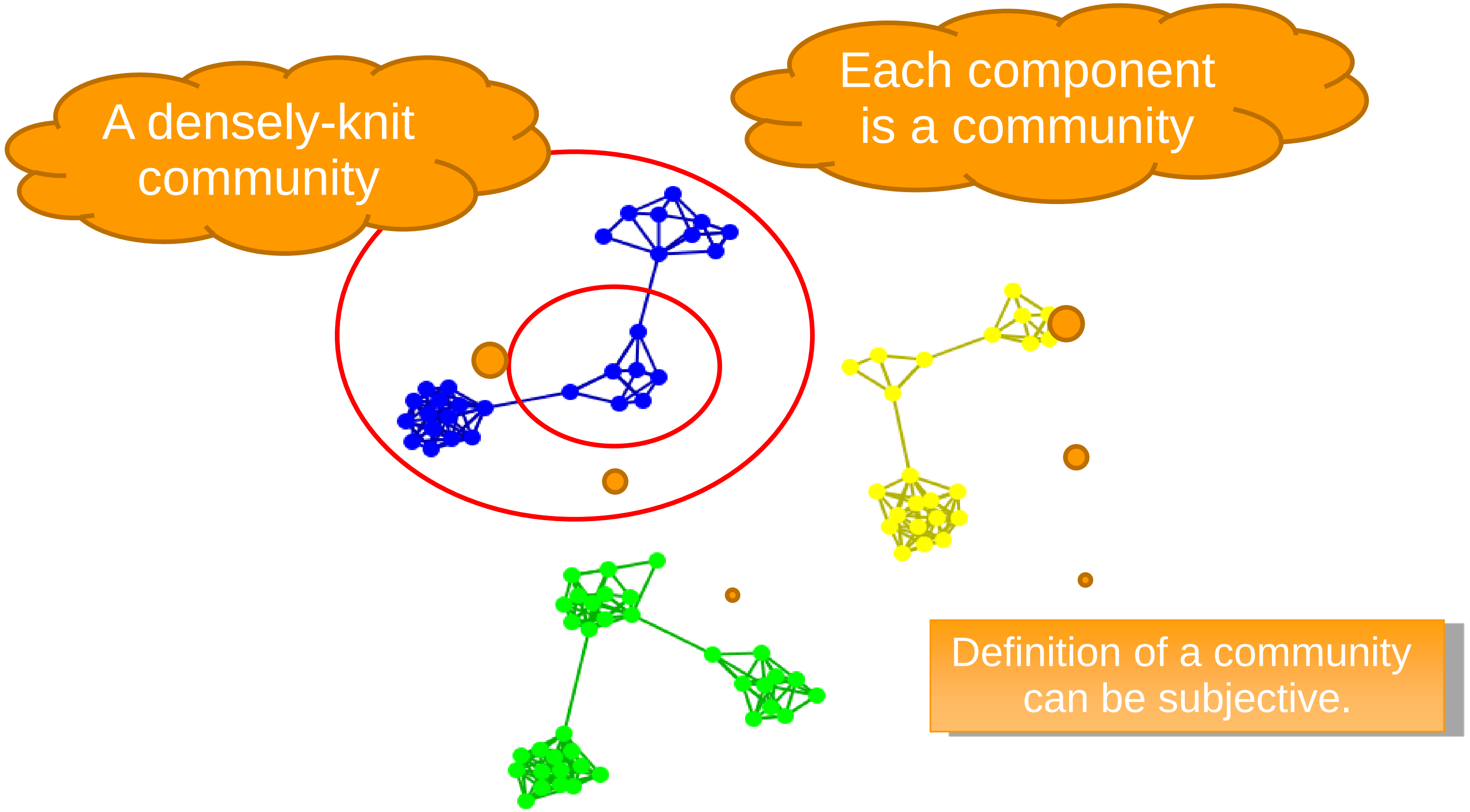
Network communities are groups of vertices similar to each other.



- Community detection is an assignment of vertices to communities.
- Non-overlapping communities (every vertex belongs to a single group)

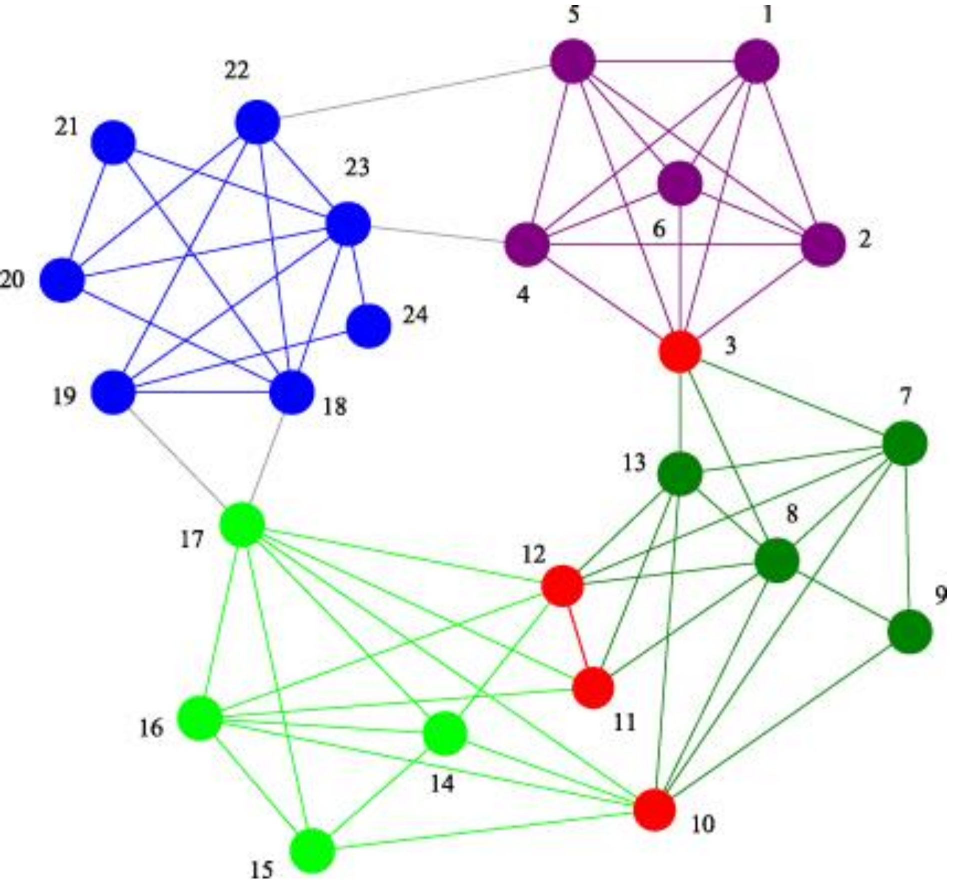
Community Identification/Clustering

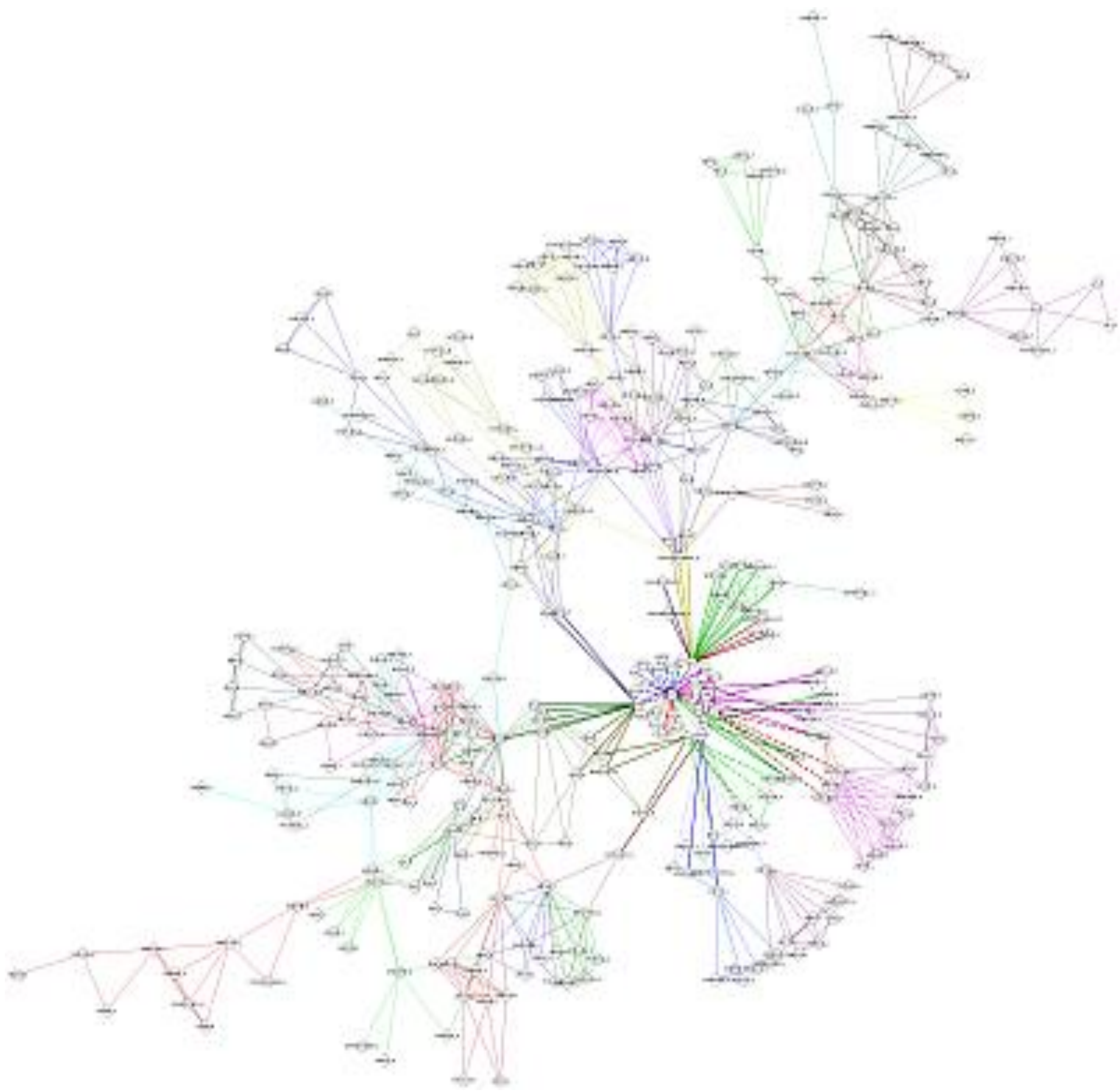
- Groups of nodes that are densely connected amongst themselves while being sparsely connected to the rest of the network



Definition of a community can be subjective.

Might not be easy to see through ...





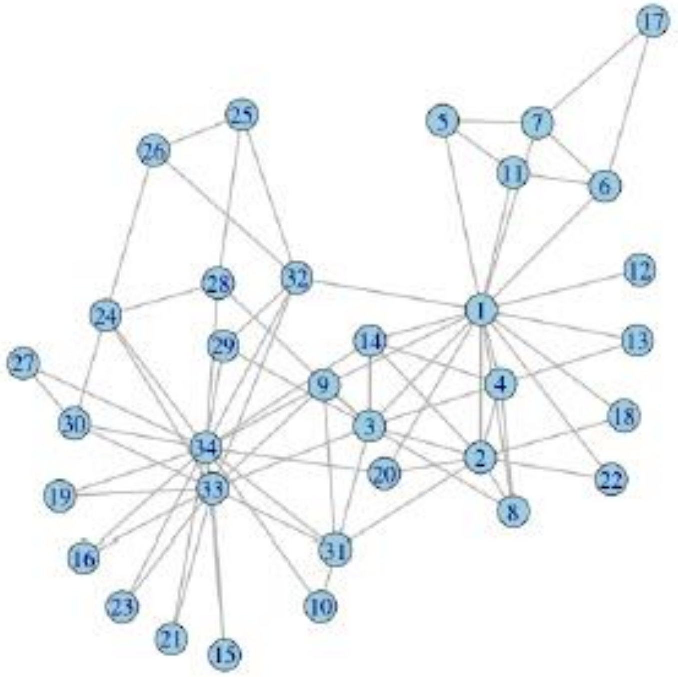
Computational Metods

- Agglomerative
  - make an empty graph (N nodes, 0 edges)
  - add edges into empty graph maximizing something in original network
- Divisive
  - cut edges in prescribed order until communities separate
- Spectral
  - split graph based on eigenvalues/eigenvectors of Graph Laplacian

Network communities

Definition

Network communities are groups of vertices similar to each other.

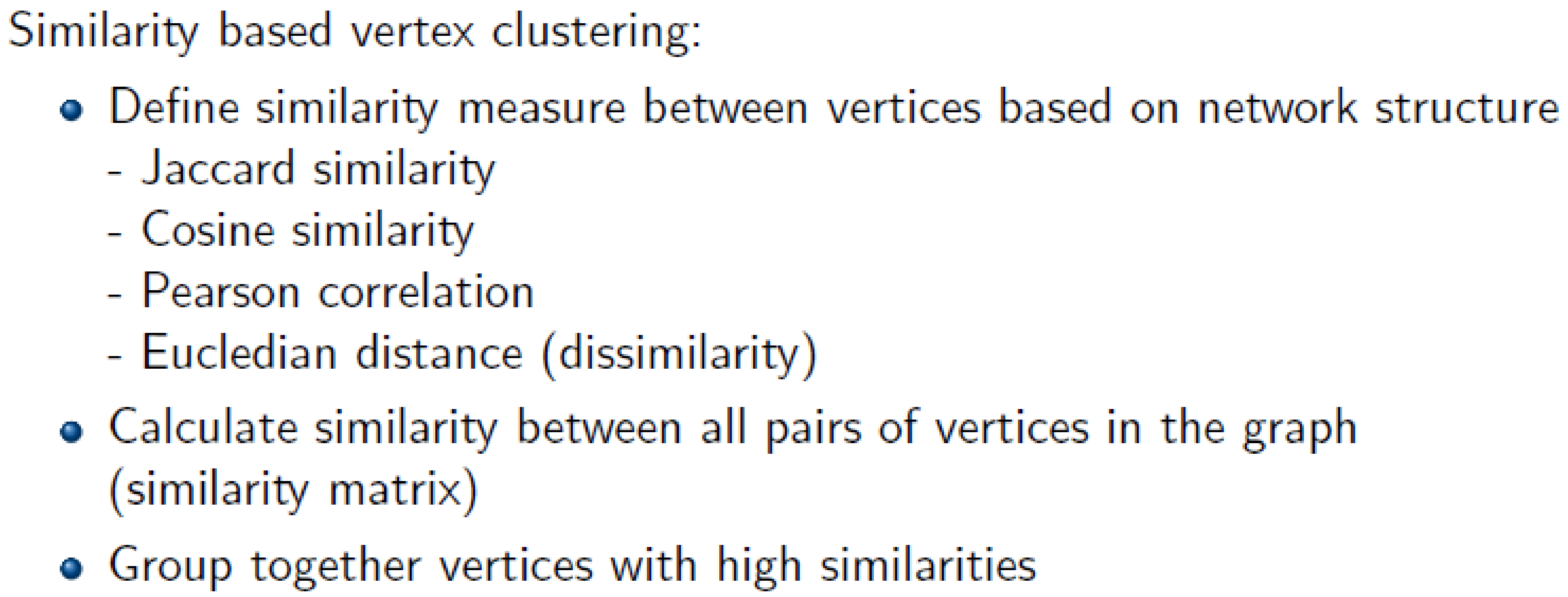


- Community detection is an assignment of vertices to communities.
- Non-overlapping communities (every vertex belongs to a single group)

Similarity Measures

- Choosing (dis)similarity measures – a critical step in community finding/clustering
- Recall that the goal is to group together "similar" data – but what does this mean?
- No single answer – it depends on what we want to find or emphasize in the data; this is one reason why clustering is an "art"
- The similarity measure is often more important than the clustering algorithm used – don't overlook this choice!

Similarity based clustering

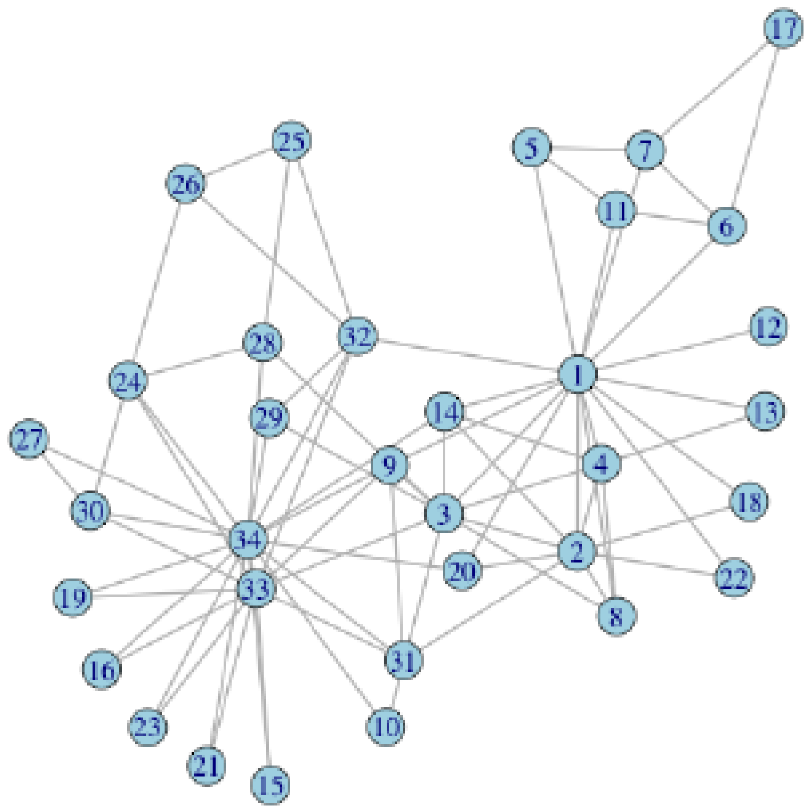


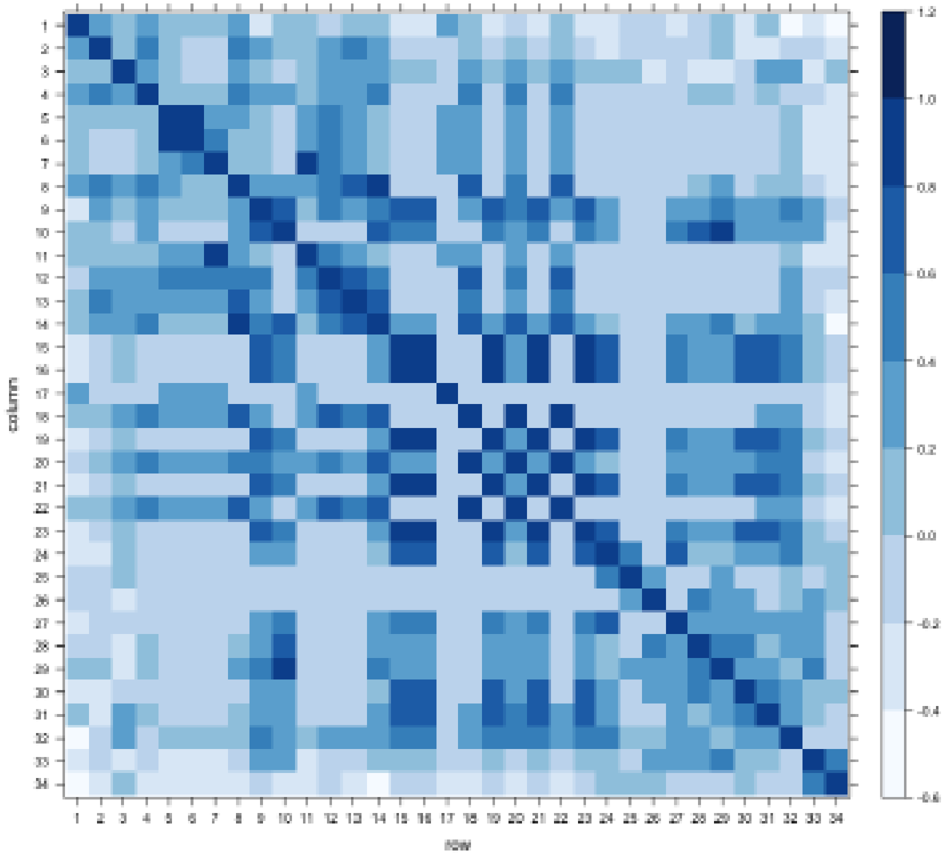
Similarity based vertex clustering:

- Define similarity measure between vertices based on network structure
  - Jaccard similarity
  - Cosine similarity
  - Pearson correlation
  - Eucledian distance (dissimilarity)
- Calculate similarity between all pairs of vertices in the graph (similarity matrix)
- Group together vertices with high similarities

Similarity matrix

Zachary karate club

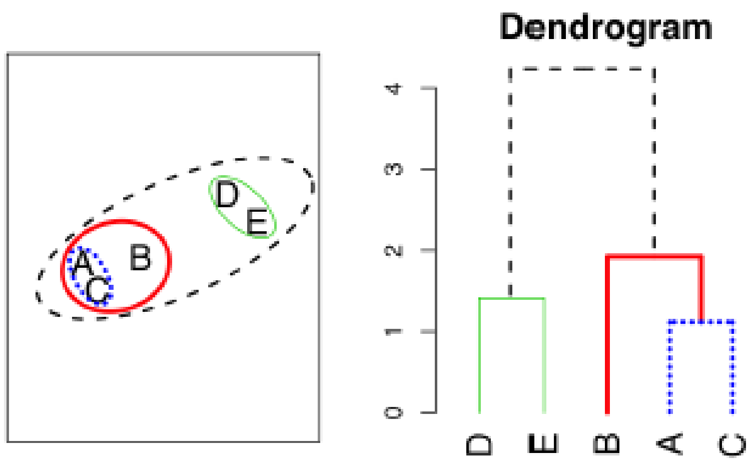




Hierarchical clustering

Agglomerative clustering:

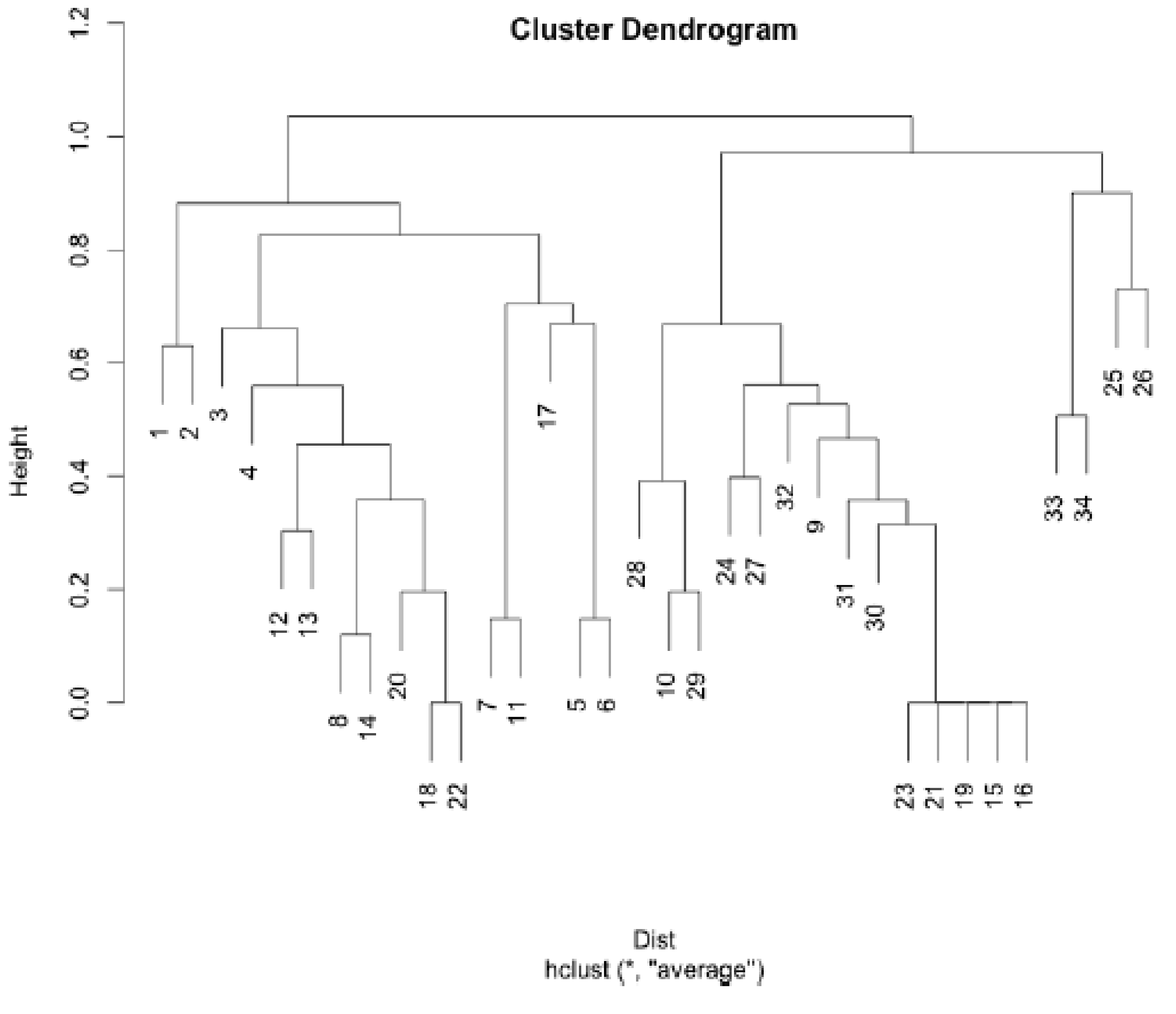
- Assign each vertex to a group of its own
- Find two groups with the highest similarity and join them in a single group
- Calculate similarity between groups:
  - single-linkage clustering (most similar in the group)
  - complete-linkage clustering (least similar in the group)
  - average-linkage clustering (mean similarity between groups)
- Repeat until all joined into single group



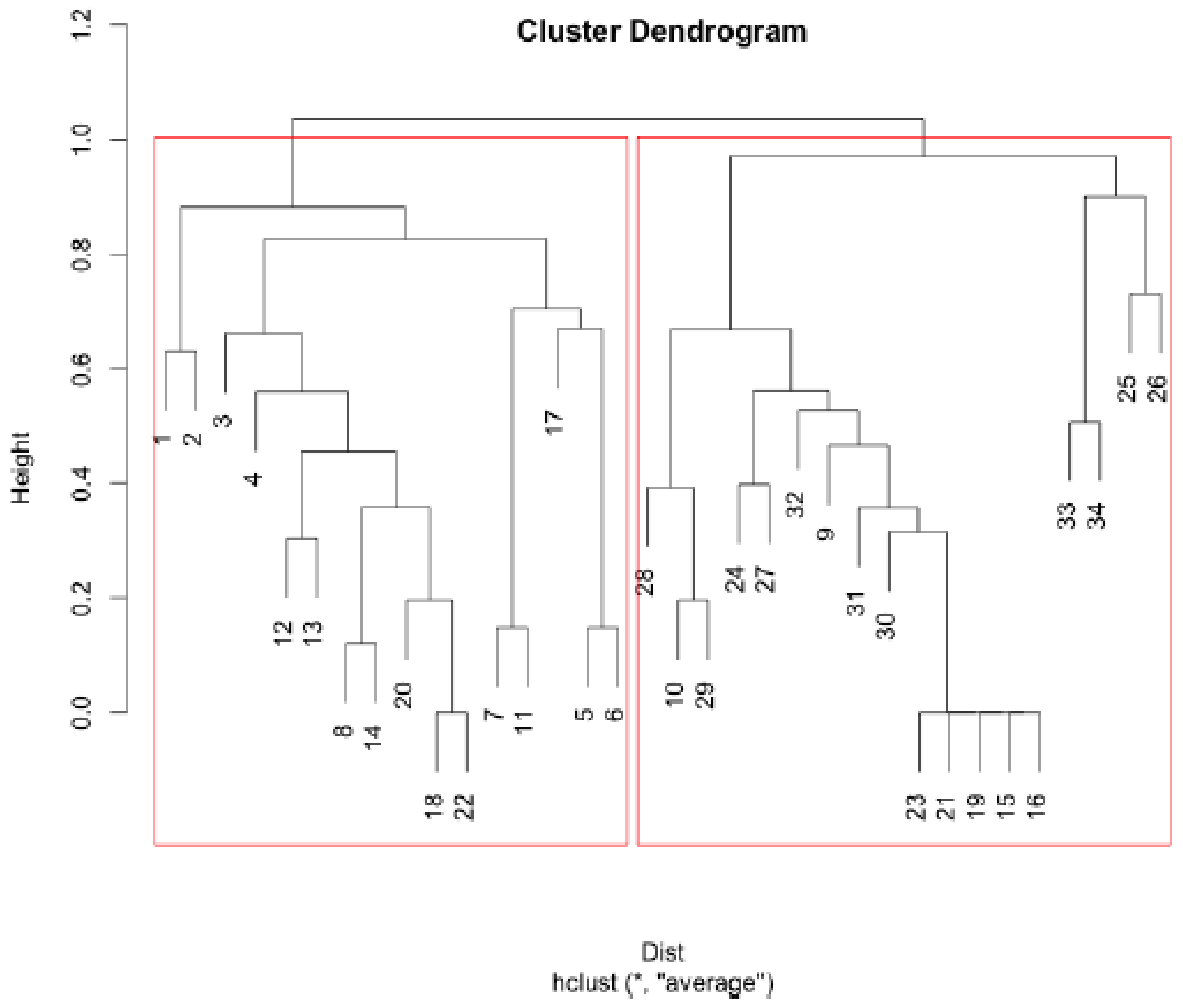
Agglomerative Method

- Start with every data point in a separate cluster
- Keep merging the most similar pairs of data points/clusters until we have one big cluster left
- This is called a bottom-up or agglomerative method

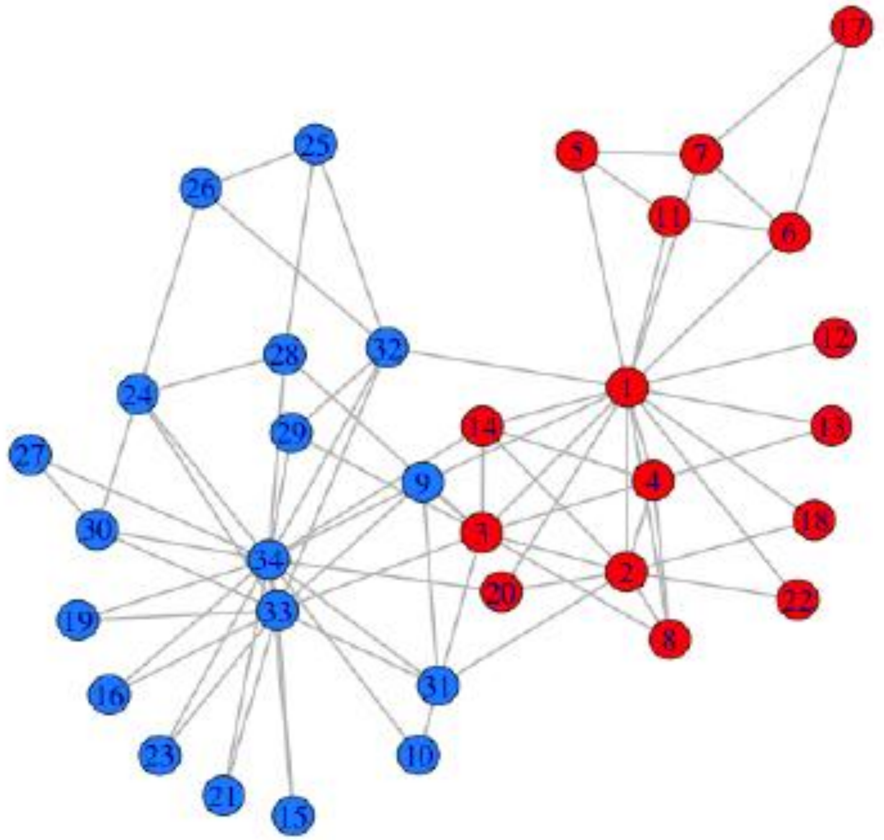
Hierarchical clustering



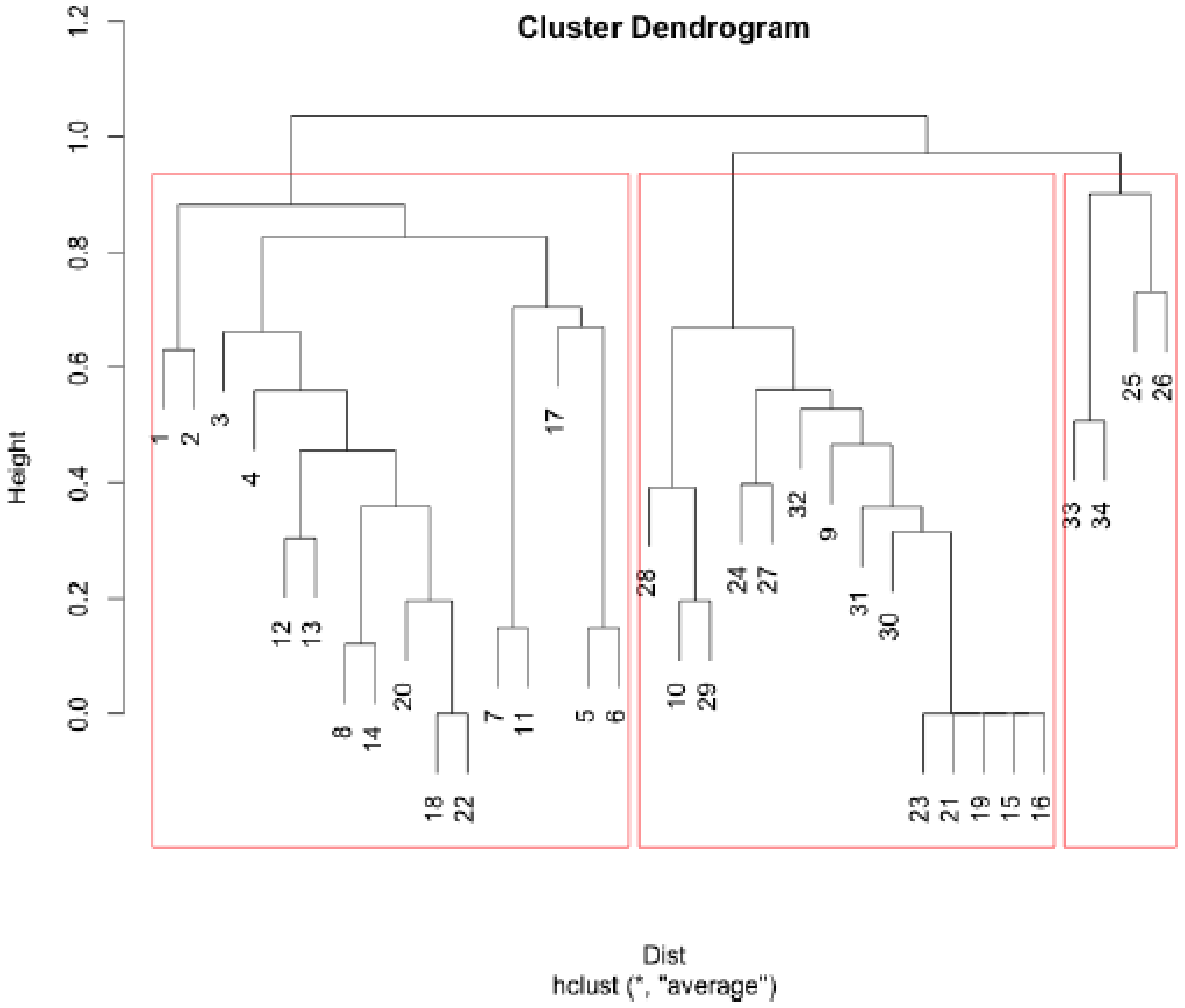
Hierarchical clustering



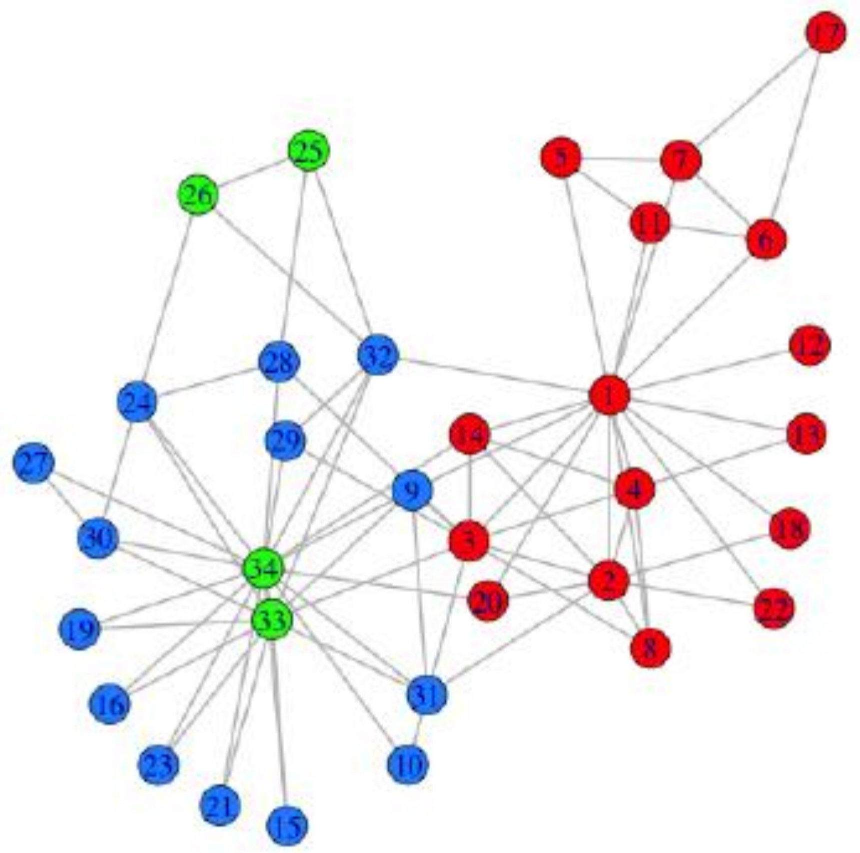
Hierarchical clustering



Hierarchical clustering



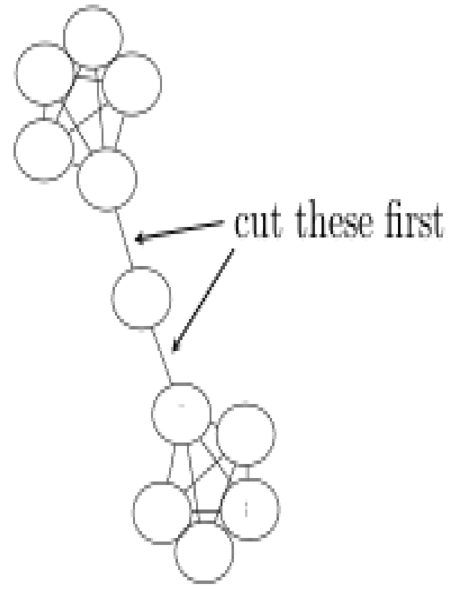
Hierarchical clustering

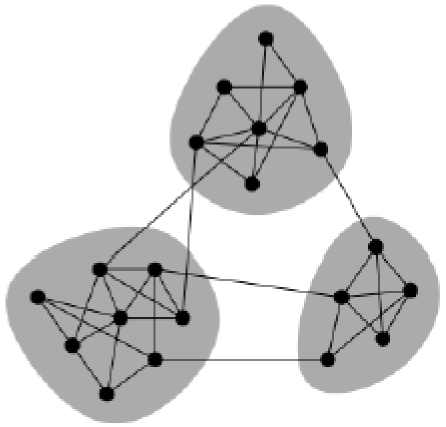


Community Detection Algorithm

Divisive Method (Newman-Girvan Algo)

- Calculate the edge betweenness for all edges in the network.
- Remove the edge with the highest betweenness.
- Recalculate betweennesses for all edges affected by the removal.
- Repeat until no edges remain.





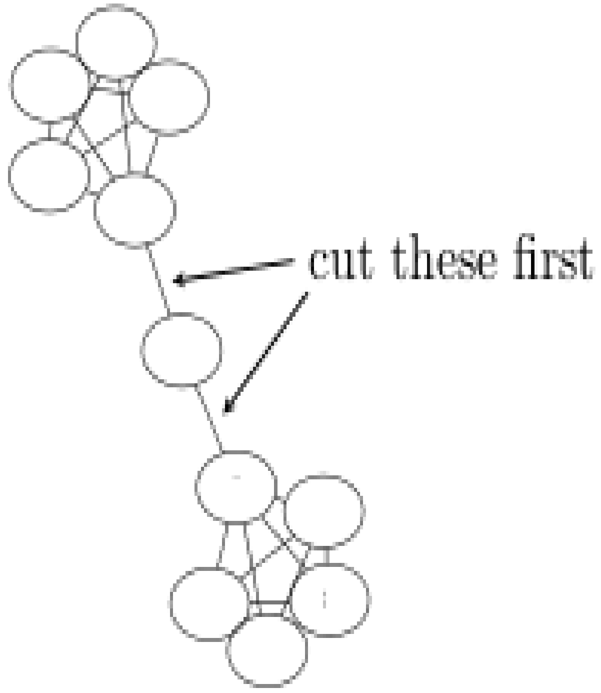
Community Detection Algorithm

Divisive Method

Edge betweenness

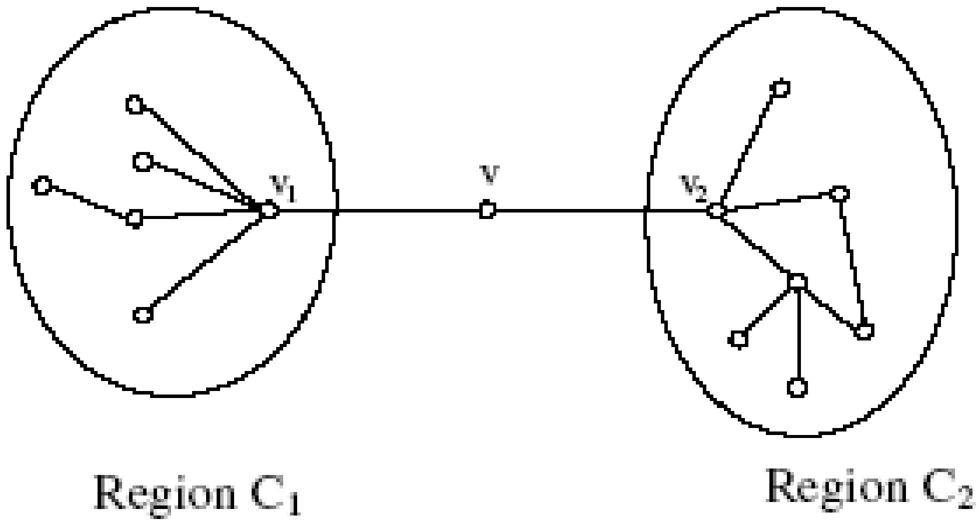
betweenness(e_{ij}) = number of times e_{ij} appears in all shortest paths

High betweenness edges are more "central"



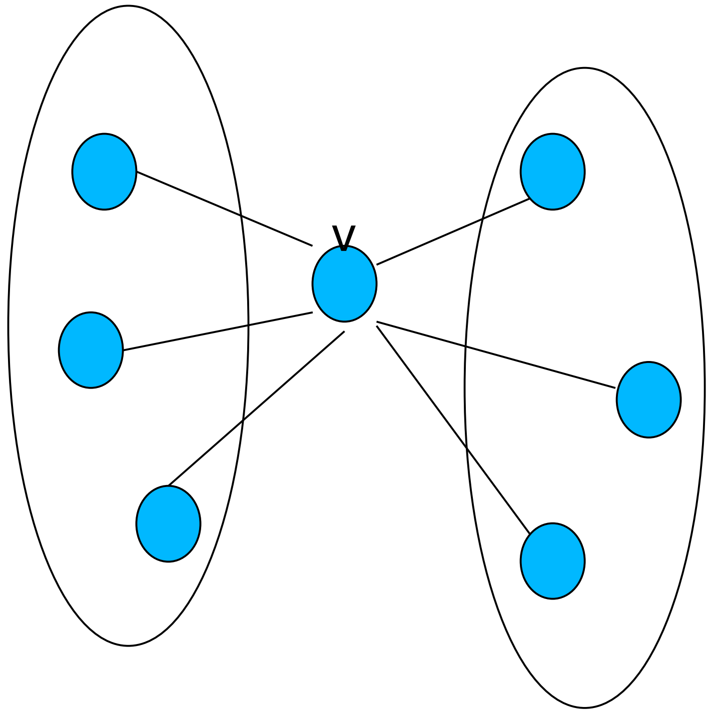
Betweenness Centrality

- Tries to determine how important is a node in a network
- Degree of a node doesn't only determine its importance in the network – do you agree???
- The node can be on a *bridge* centrally between two regions of the network!!



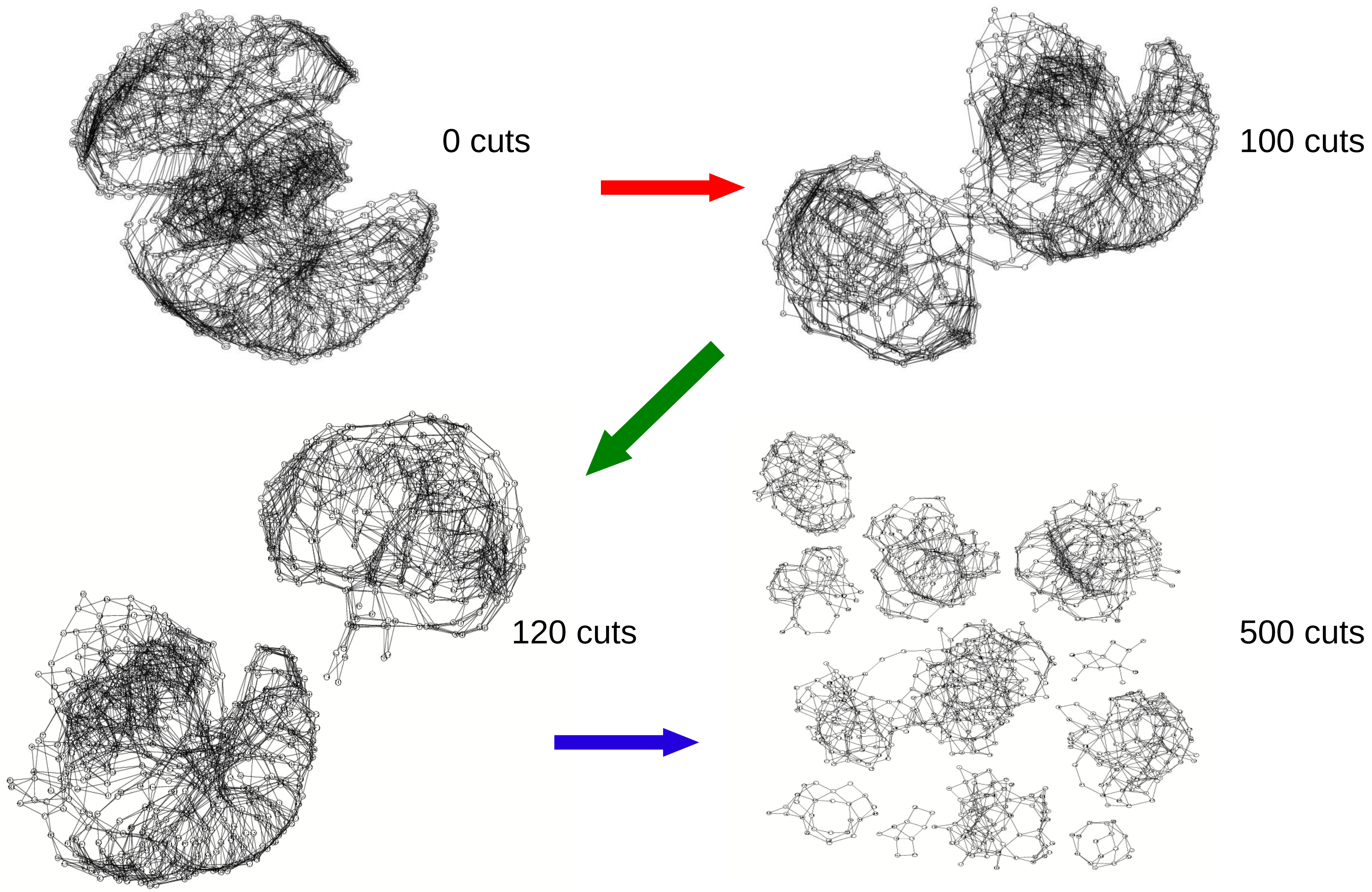
Betweenness Centrality

- Centrality of *v*: Ratio: the number of shortest paths that pass through v Vs total number of shortest paths from node s to node t.

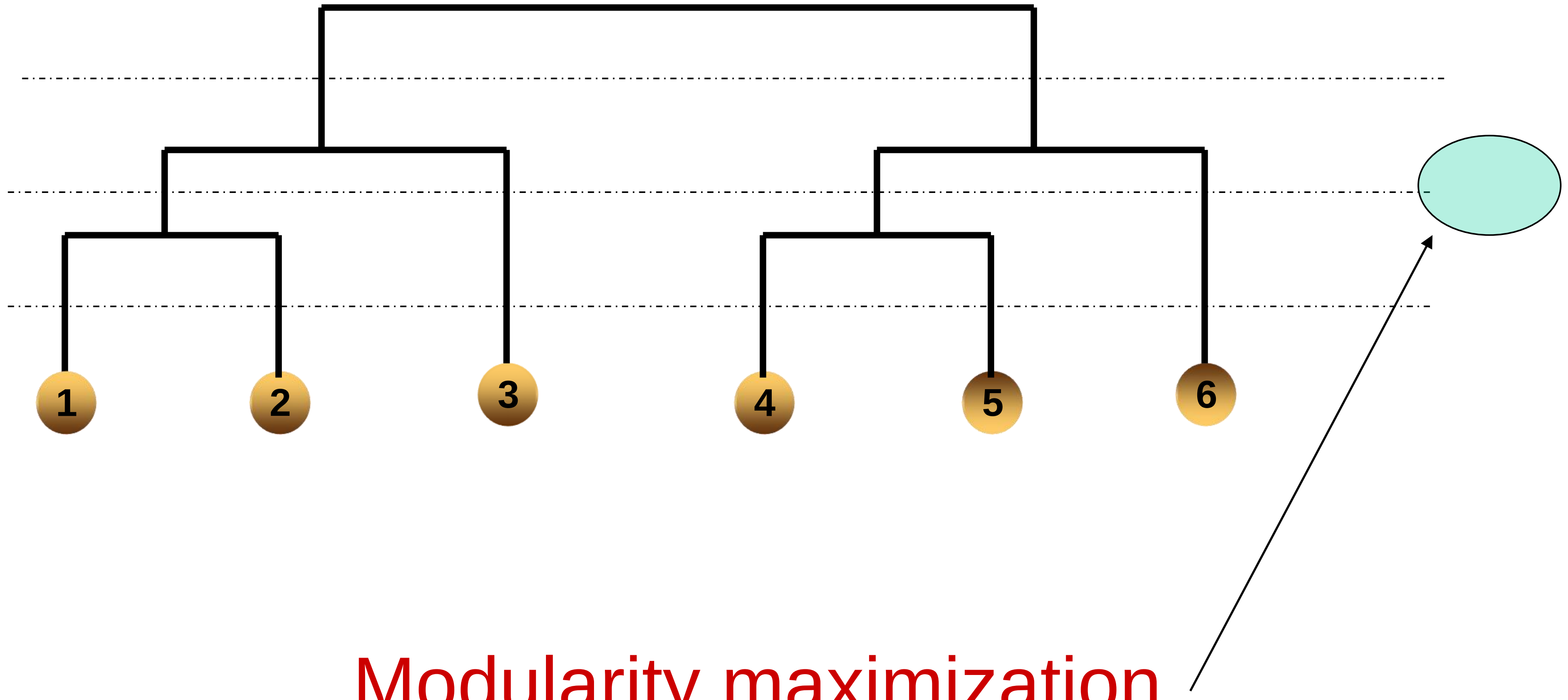


$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

In Execution



Community Detection



Modularity maximization

How good is a community

- Communities are dense compared to random case
- Measured in terms of modularity
- Total number of in-community edges – expected number of edges if there is no community structure

Modularity

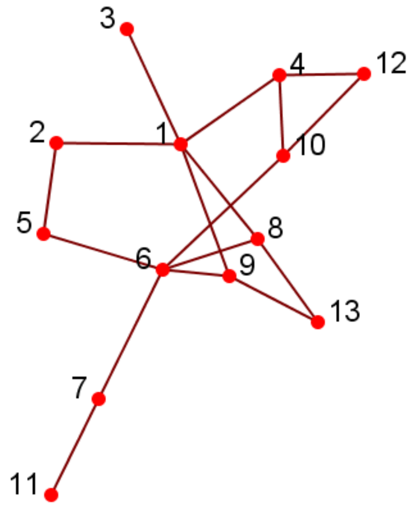
Modularity measures the group interactions compared with the expected random connections in the group

Apriori communities

In a network with m edges, for two nodes with degree d_i and d_j , expected random connections between them are

$$\boxed{d_i d_j / 2m}$$

NULL model



The interaction utility in a group:

$$\sum_{i \in C, j \in C} A_{ij} - d_i d_j / 2m$$

Modularity

$$\frac{1}{2m} \sum_{C} \sum_{i \in C, j \in C} A_{ij} - d_i d_j / 2m$$

Expected Number of edges between 6 and 9 is
5*3/(2*17) = 15/34

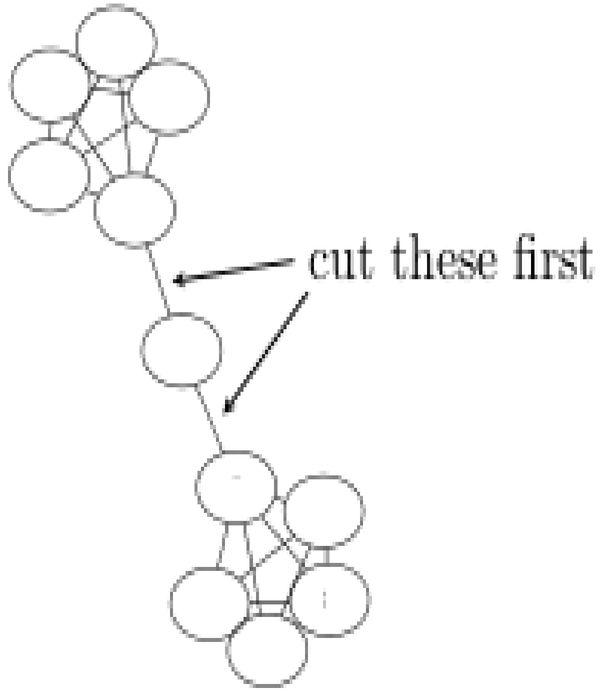
Bounds between [-0.5: +1]

Community Detection Algorithm

Edge betweenness

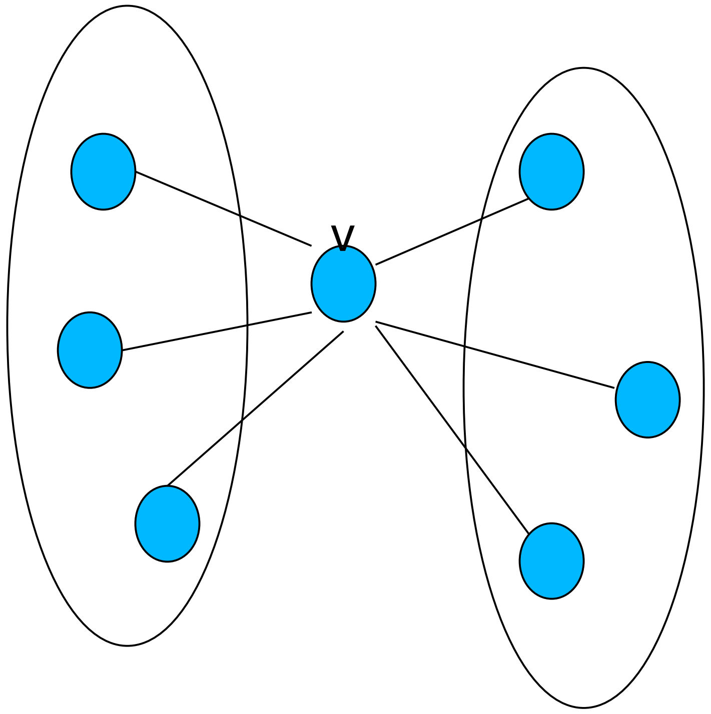
betweenness(e_{ij}) = number of times e_{ij} appears in all shortest paths

High betweenness edges are more "central"



Betweenness Centrality

- Centrality of *v*: Ratio: the number of shortest paths that pass through v Vs total number of shortest paths from node s to node t.



$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

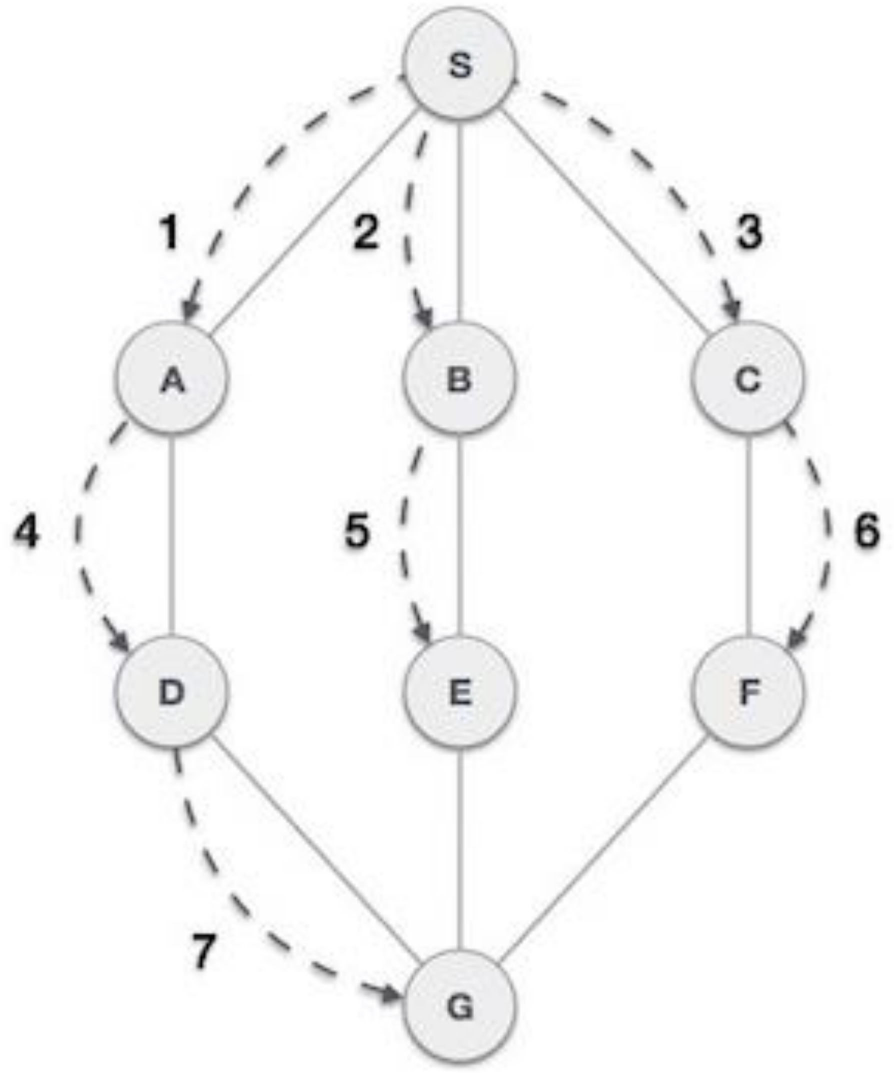
Computing edge betweenness

- Finding shortest path between node pairs
  - n nodes and m edges
- **Floyd–Warshall algorithm – O(n^3)**
- Compute shortest path between (s,d) using BFS (O(m))

Repeat for all pairs O(mn^2)

- Case 1: Only one shortest path between any node pair
- BFS tree shows those paths

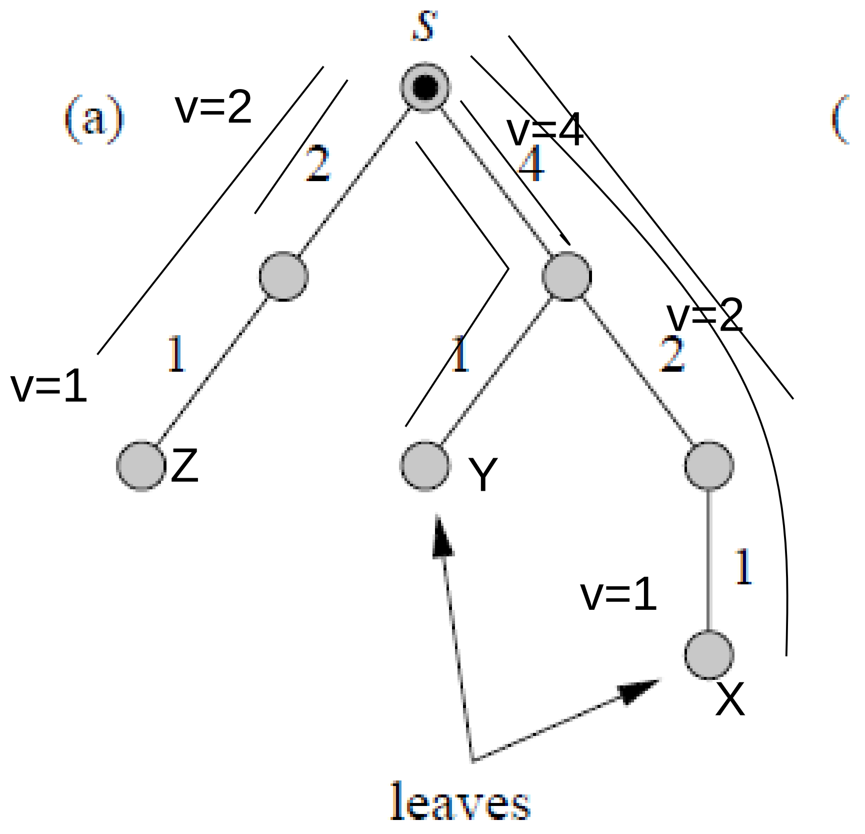
Breadth first search



Construct BFS tree

Computing edge betweenness

Case 1: Only one shortest path between any node pair



Use this tree to calculate the contribution of each edge to betweenness

Step 1: Find the leaves => No shortest paths to other nodes pass through them.

Assign score 1 to each such leave edge

Step 2: Starting from farthest edges, walk upwards

Assign score v to each edge
1+ (sum of the scores on the neighboring edges immediately below it)

Result score v for each edge:
Betweenness counts of the paths from source S

Repeat this process for all S node and sum up the scores for each edge

Results full betweenness score for shortest paths of all pairs

Computing edge betweenness

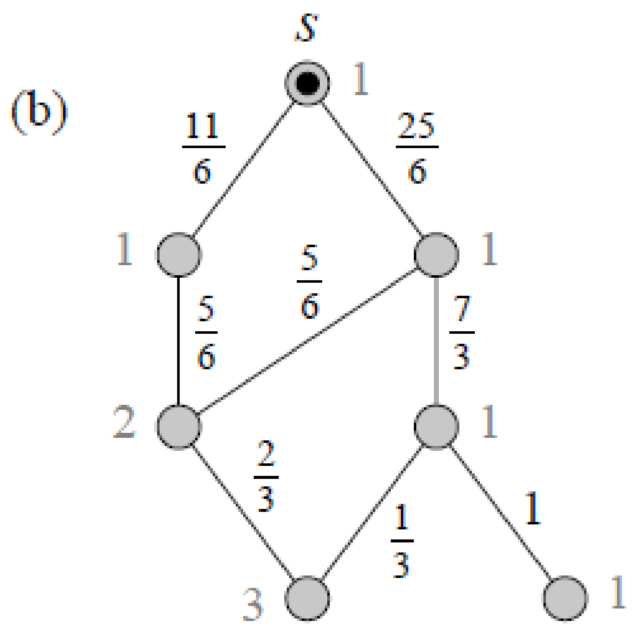
The breadth-first search and the process of working up through the tree both take worst-case time O(m)

there are n vertices total, so the entire calculation takes time O(mn) as claimed.

Computing edge betweenness

Case 2: Multiple shortest paths between any node pair

Key idea: If there are three shortest paths between (a,b) passing through an edge e, each will contribute 1/3 to edge betweenness of e

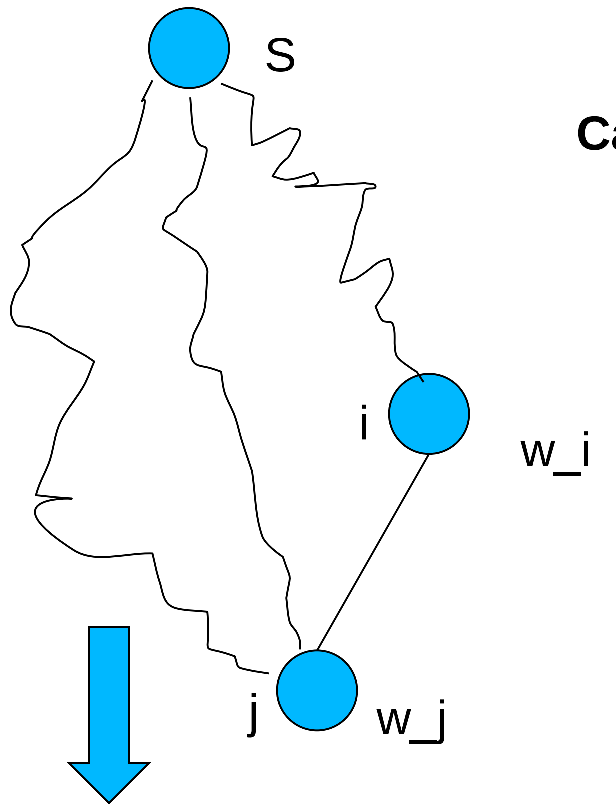


1. The initial vertex s is given distance $d_s = 0$ and a weight $w_s = 1$.
2. Every vertex i adjacent to s is given distance $d_i = d_s + 1 = 1$, and weight $w_i = w_s = 1$.
3. For each vertex j adjacent to one of *those* vertices i we do one of three things:
   (a) If j has not yet been assigned a distance, it is assigned distance $d_j = d_i + 1$ and weight $w_j = w_i$.
   (b) If j has already been assigned a distance and $d_j = d_i + 1$, then the vertex's weight is increased by w_i, that is $w_j \leftarrow w_j + w_i$.
   (c) If j has already been assigned a distance and $d_j < d_i + 1$, we do nothing.
4. Repeat from step 3 until no vertices remain that have assigned distances but whose neighbors do not have assigned distances.

First perform the BFS on source S and obtain the graph

The weight on a vertex i represents the number of distinct paths from S to i

Computing edge betweenness



Calculation of edge weight (i,j)

S to i : w_i different paths

Paths: S to j via i, results w_i different paths

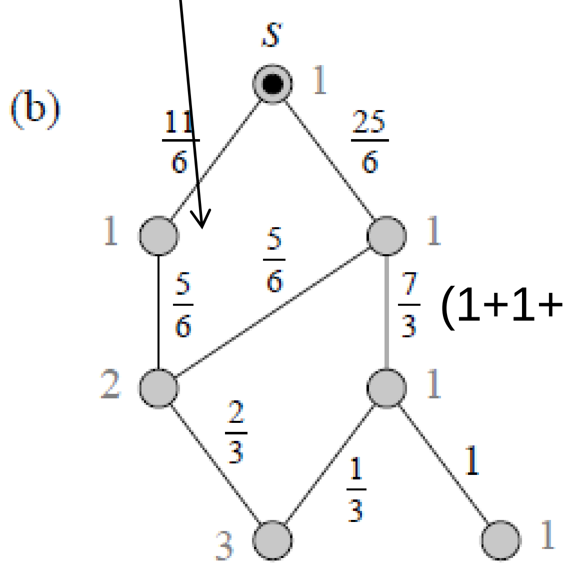
So contribution of (i,j)=w_i/w_j

fraction of shortest paths from j through i to s

w_j different paths ----- contributes to (i,j) = 1/w_j

Computing edge betweenness

(1+2/3)*1/2=5/6



(1+1+1/3)*1

1. Find every "leaf" vertex t, i.e., a vertex such that no paths from s to other vertices go though t.

2. For each vertex i neighboring t assign a score to the edge from t to i of w_i/w_t.

3. Now, starting with the edges that are farthest from the source vertex s—lower down in a diagram such as Fig. 4b—work up towards s. To the edge from vertex i to vertex j, with j being farther from s than i, assign a score that is 1 plus the sum of the scores on the neighboring edges immediately below it (i.e., those with which it shares a common vertex), all multiplied by w_i/w_j.

4. Repeat from step 3 until vertex s is reached.

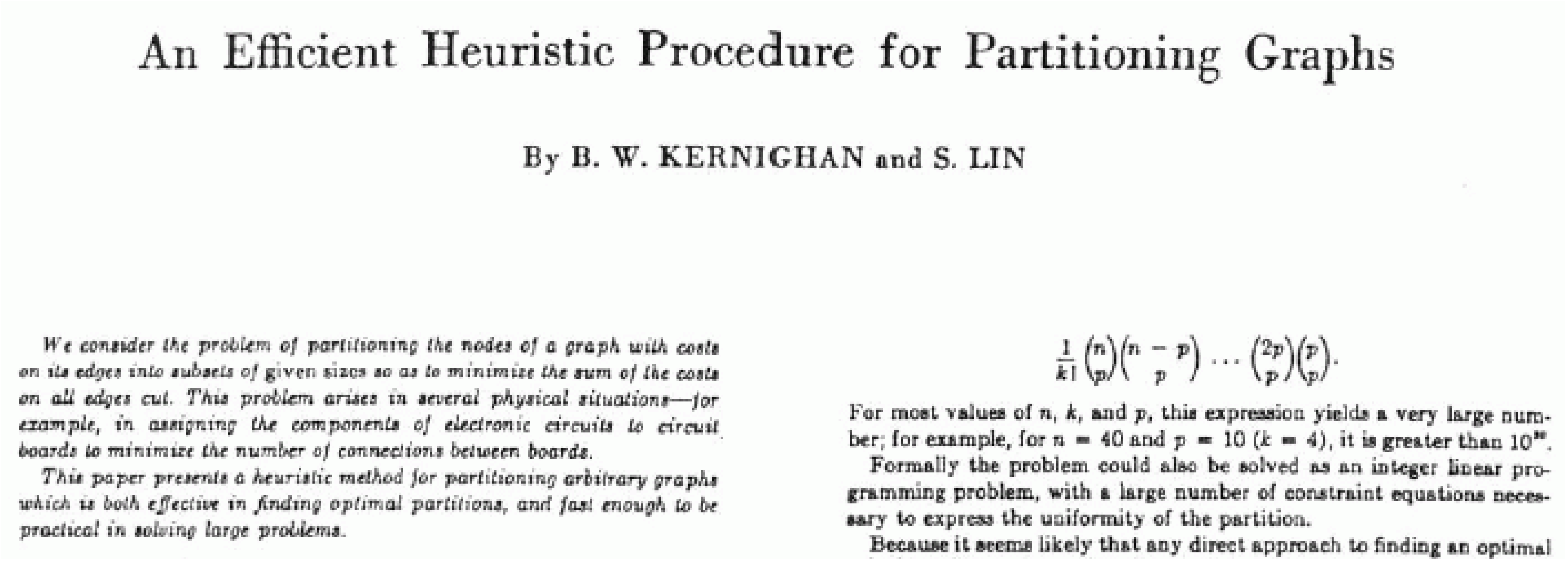
(1+ sum of the scores on the neighboring edges immediately below it)*w_i/w_j

Computing edge betweenness

Now repeating this process for all n source vertices s and summing the resulting scores on the edges gives us the total betweenness for all edges in time $O(mn)$.

Kernighan and Lin heuristic

"An Efficient Heuristic Procedure for Partitioning Graphs" B. W. Kernighan and S. Lin, The Bell System Technical Journal, 49(2):291-307, 1970



An Efficient Heuristic Procedure for Partitioning Graphs

By B. W. KERNIGHAN and S. LIN

We consider the problem of partitioning the nodes of a graph with costs on its edges into subsets of given sizes so as to minimize the sum of the costs on all edges cut. This problem arises in several physical situations—for example, in assigning the components of electronic circuits to circuit boards to minimize the number of connections between boards.

This paper presents a heuristic method for partitioning arbitrary graphs which is both effective in finding optimal partitions, and fast enough to be practical in solving large problems.

$$\frac{1}{k!}\binom{n}{p}\binom{n-p}{p}\cdots\binom{2p}{p}\binom{p}{p}.$$

For most values of n, k, and p, this expression yields a very large number; for example, for $n = 40$ and $p = 10$ ($k = 4$), it is greater than $10^{[illegible]}$.

Formally the problem could also be solved as an integer linear programming problem, with a large number of constraint equations necessary to express the uniformity of the partition.

Because it seems likely that any direct approach to finding an optimal

Idea of KL Algorithm

Start with any initial partition X and Y.

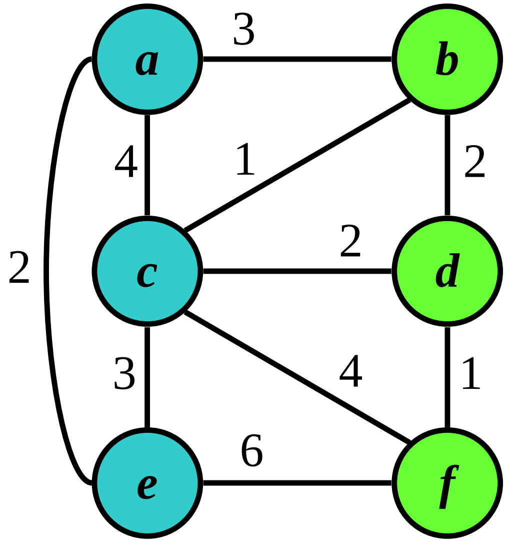
A pass or iteration means exchanging each vertex $A \in X$ with each vertex $B \in Y$ exactly once:

```
1. For i := 1 to n do
      From the unlocked (unexchanged) vertices,
        choose a pair (A,B) s.t. gain(A,B) is largest.
      Exchange A and B. Lock A and B.
      Let $g_i$ = gain(A,B).
2. Find the k s.t. $G=g_1+...+g_k$ is maximized.
3. Switch the first k pairs.
```

Repeat the pass until there is no improvement (G=0).

The gain function = (number of edges that lie within the two groups) – (the number of edges that lie between them)

Kernighan-Lin Algorithm (1)



Given:

Initial weighted graph *G* with

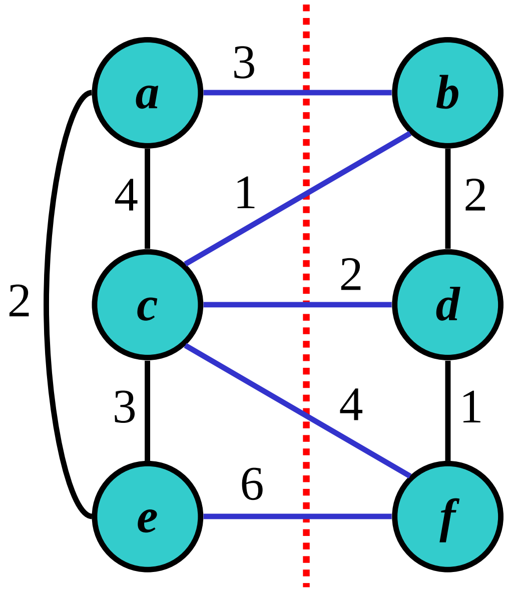
***V*(*G*) = { *a*, *b*, *c*, *d*, *e*, *f* }**

Start with any partition of V(G) into X and Y, say

***X* = { *a*, *c*, *e* }**

***Y* = { *b*, *d*, *f* }**

KL algorithm (2a)



cut-size = 3+1+2+4+6 = 16

$X = \{ a, c, e \}$
$Y = \{ b, d, f \}$

Compute the gain values of moving node x to the others set:

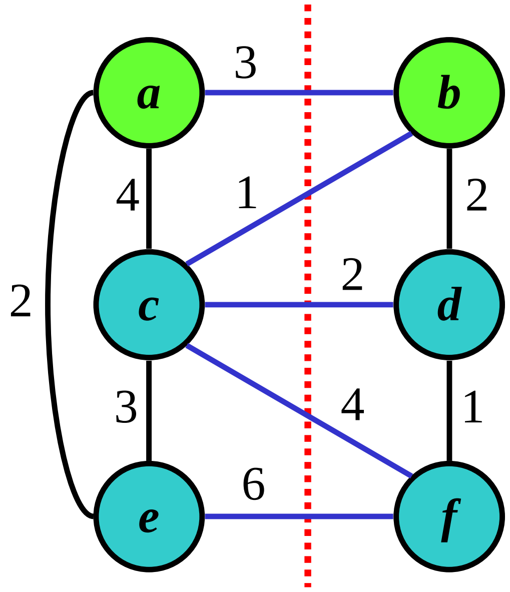
$G_x = E_x - I_x$

E_x **= cost of edges connecting node x with the other group (exter)**

I_x **= cost of edges connecting node x within its own group (intra)**

$G_a = E_a - I_a = -3 \quad (= 3 - 4 - 2)$
$G_c = E_c - I_c = \quad 0 \quad (= 1 + 2 + 4 - 4 - 3)$
$G_e = E_e - I_e = +1 \quad (= 6 - 2 - 3)$
$G_b = E_b - I_b = +2 \quad (= 3 + 1 - 2)$
$G_d = E_d - I_d = -1 \quad (= 2 - 2 - 1)$
$G_f = E_f - I_f = +9 \quad (= 4 + 6 - 1)$

KL algorithm (2b)



Cost saving when exchanging *a* and *b* is essentially $G_a + G_b$

However, the cost saving 3 of the direct edge was counted twice. But this edge still connects the two groups

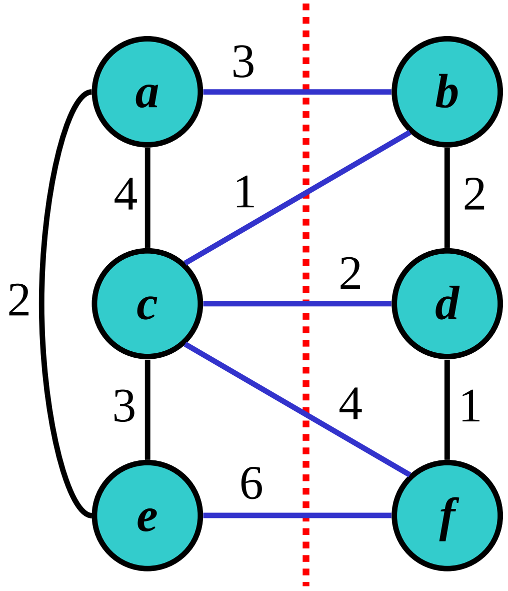
Hence, the real "gain" (i.e. cost saving) of this exchange is $g_{ab} = G_a + G_b - 2c_{ab}$

$X = \{ a, c, e \}$
$Y = \{ b, d, f \}$

$G_a = E_a - I_a = -3 \;\; (= 3 - 4 - 2)$
$G_b = E_b - I_b = +2 \;\; (= 3 + 1 - 2)$
$g_{ab} = G_a + G_b - 2c_{ab} = -7 \; (= -3 + 2 - 2 \cdot 3)$

KL algorithm (3)

$G_a = -3 \quad G_b = +2$
$G_c = 0 \quad G_d = -1$
$G_e = +1 \quad G_f = +9$



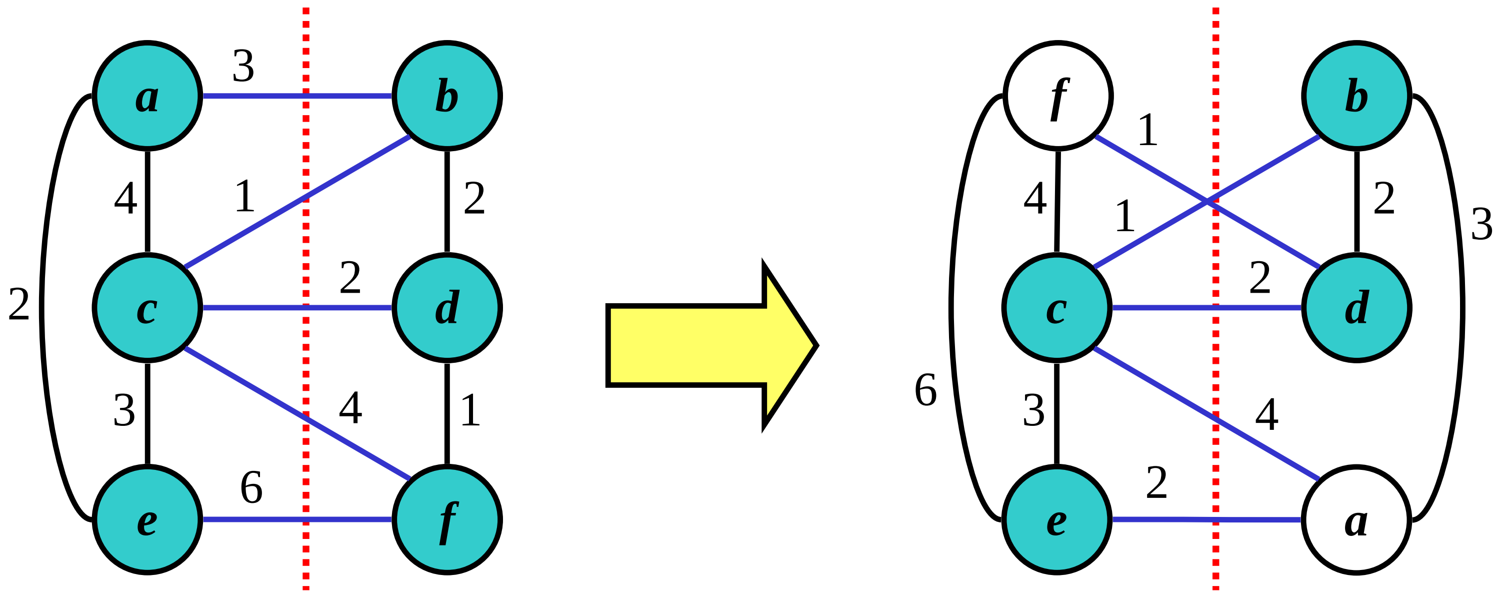
cut-size = 16

Compute all the gains

$g_{ab} = G_a + G_b - 2w_{ab} = -3 + 2 - 2\cdot3 = -7$
$g_{ad} = G_a + G_d - 2w_{ad} = -3 - 1 - 2\cdot0 = -4$
$g_{af} = G_a + G_f - 2w_{af} = -3 + 9 - 2\cdot0 = +6$
$g_{cb} = G_c + G_b - 2w_{cb} = 0 + 2 - 2\cdot1 = 0$
$g_{cd} = G_c + G_d - 2w_{cd} = 0 - 1 - 2\cdot2 = -5$
$g_{cf} = G_c + G_f - 2w_{cf} = 0 + 9 - 2\cdot4 = +1$
$g_{eb} = G_e + G_b - 2w_{eb} = +1 + 2 - 2\cdot0 = +3$
$g_{ed} = G_e + G_d - 2w_{ed} = +1 - 1 - 2\cdot0 = 0$
$g_{ef} = G_e + G_f - 2w_{ef} = +1 + 9 - 2\cdot6 = -2$

Pair with maximum gain

KL algorithm (4)



cut-size = 16

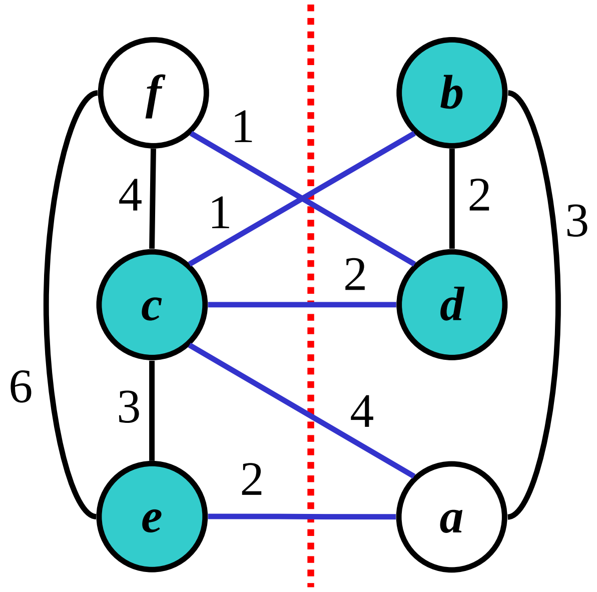
cut-size = 16 – 6 = 10

Exchange nodes *a* and *f*

Then lock up nodes *a* and *f*

$$g_{af} = G_a + G_f - 2c_{af} = -3 + 9 - 2 \cdot 0 = +6$$

KL algorithm (5)



cut-size = 10

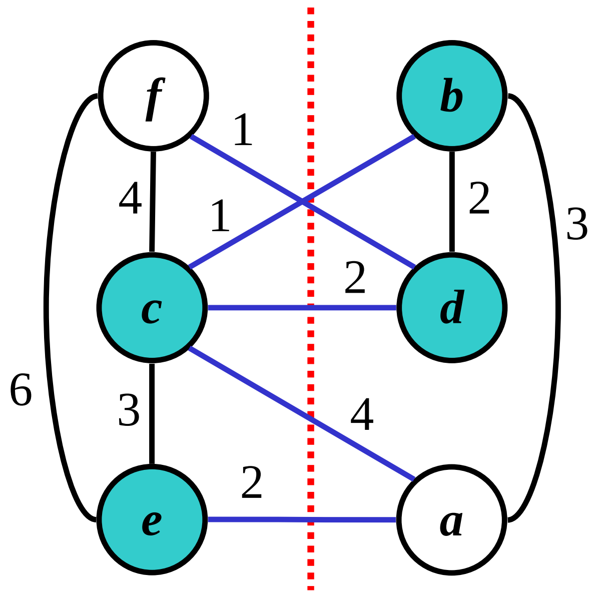
$X' = \{ c, e \}$
$Y' = \{ b, d \}$

$G_a = -3$	$G_b = +2$
$G_c = 0$	$G_d = -1$
$G_e = +1$	$G_f = +9$

Update the G-values of unlocked nodes

$$G'_c = G_c + 2c_{ca} - 2c_{cf} = 0 + 2(4 - 4) = 0$$
$$G'_e = G_e + 2c_{ea} - 2c_{ef} = 1 + 2(2 - 6) = -7$$
$$G'_b = G_b + 2c_{bf} - 2c_{ba} = 2 + 2(0 - 3) = -4$$
$$G'_d = G_d + 2c_{df} - 2c_{da} = -1 + 2(1 - 0) = 1$$

KL algorithm (6)



cut-size = 10

$X' = \{ c, e \}$
$Y' = \{ b, d \}$

$G'_c = 0 \quad G'_b = -4$
$G'_e = -7 \quad G'_d = +1$

Compute the gains

$g'_{cb} = G'_c + G'_b - 2c_{cb} = 0 - 4 - 2 \cdot 1 = -6$

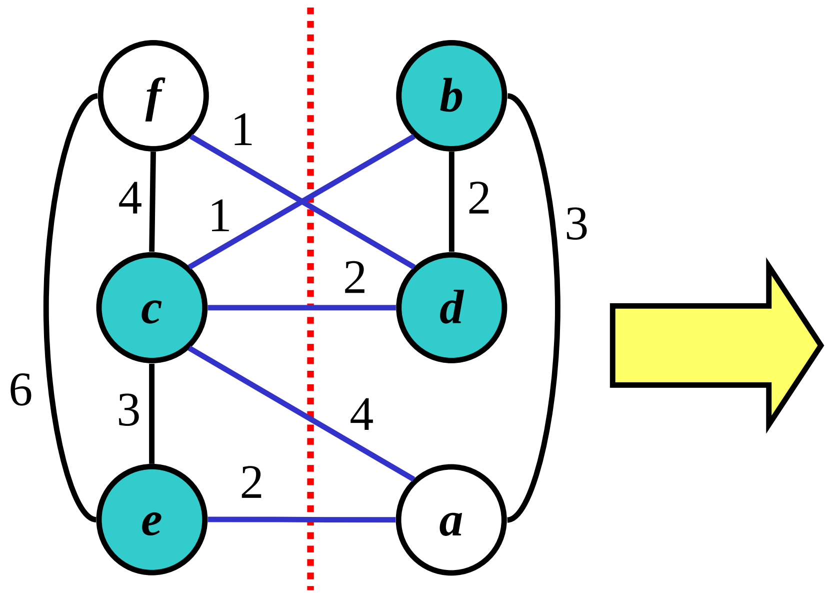
$g'_{cd} = G'_c + G'_d - 2c_{cd} = 0 + 1 - 2 \cdot 2 = -3$

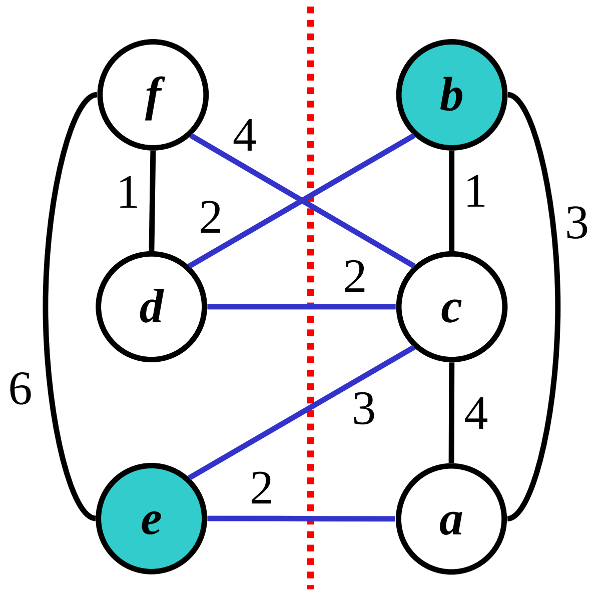
$g'_{eb} = G'_e + G'_b - 2c_{eb} = -7 - 4 - 2 \cdot 0 = -11$

$g'_{ed} = G'_e + G'_d - 2c_{ed} = -7 + 1 - 2 \cdot 0 = -6$

Pair with maximum gain (can also be neative)

KL algorithm (7)





cut-size = 10

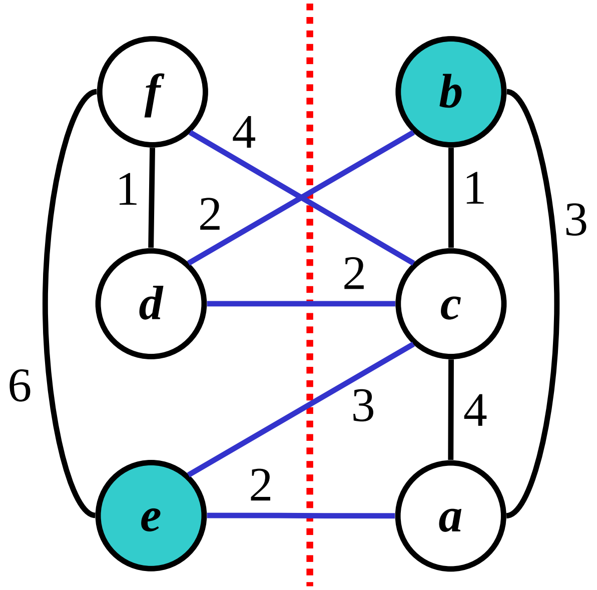
cut-size = 10 – (–3) = 13

Exchange nodes *c* and *d*

Then lock up nodes *c* and *d*

$$g'_{cd} = G'_c + G'_d - 2c_{cd} = 0 + 1 - 2\cdot 2 \quad = -3$$

KL algorithm (8)



cut-size = 13

$G'_c = 0 \quad G'_b = -4$
$G'_e = -7 \quad G'_d = +1$

$X'' = \{ e \}$
$Y'' = \{ b \}$

Update the G-values of unlocked nodes

$$G''_e = G'_e + 2c_{ed} - 2c_{ec} = -7 + 2(0 - 3) = -1$$
$$G''_b = G'_b + 2c_{bd} - 2c_{bc} = -4 + 2(2 - 1) = -2$$

Compute the gains

Pair with max. gain is (*e*, *b*) $\rightarrow$

$$g''_{eb} = G''_e + G''_b - 2c_{eb} = -1 - 2 - 2\cdot 0 = -3$$

KL algorithm (9)

Summary of the Gains…

$g = +6$

$g + g' = +6 - 3 = +3$

$g + g' + g'' = +6 - 3 - 3 = 0$

Maximum Gain = $g = +6$

Exchange only nodes *a* and *f*.

End of 1 pass.

- ***Consider this as initial state***
- ***Repeat the Kernighan-Lin.***

Idea of KL Algorithm

Start with any initial partition X and Y.

A pass or iteration means exchanging each vertex A $\in$ X with each vertex B $\in$ Y exactly once:

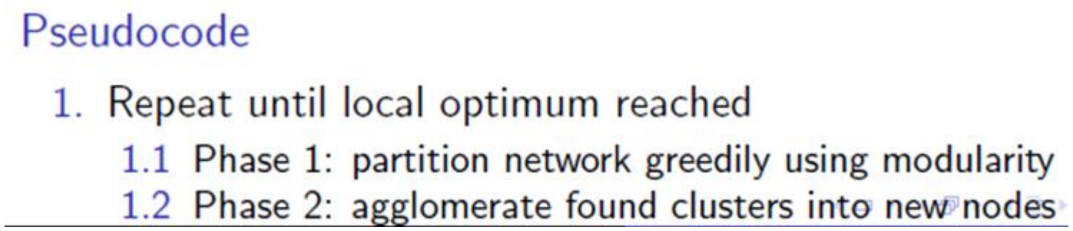
```
1. For i := 1 to n do
     From the unlocked (unexchanged) vertices,
       choose a pair (A,B) s.t. gain(A,B) is largest.
     Exchange A and B. Lock A and B.
     Let $g_i$ = gain(A,B).
2. Find the k s.t. $G=g_1+...+g_k$ is maximized.
3. Switch the first k pairs.
```

Repeat the pass until there is no improvement (G=0).

Louvain algorithm

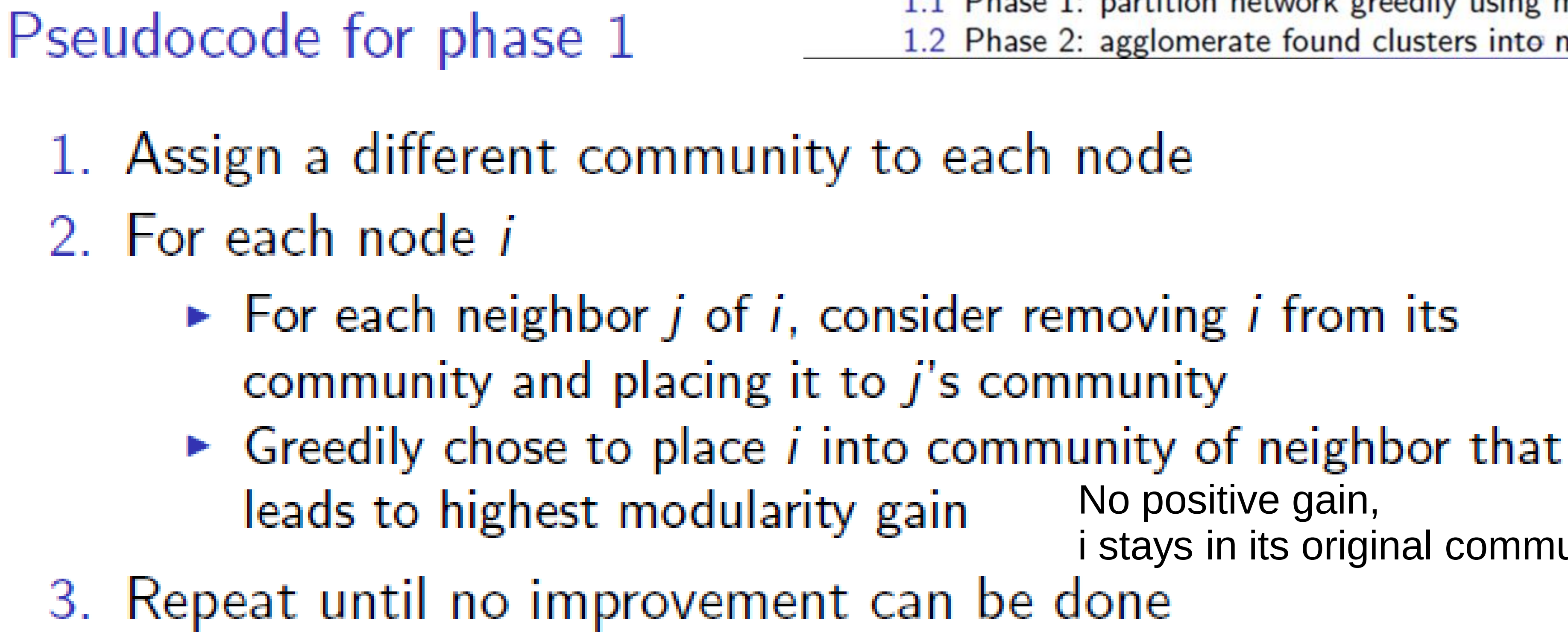
Start with the weighted network of N nodes

Introduce “pass”-----phase 1 and phase 2



Pseudocode

```
1. Repeat until local optimum reached
    1.1 Phase 1: partition network greedily using modularity
    1.2 Phase 2: agglomerate found clusters into new nodes
```



Pseudocode for phase 1

```
1. Assign a different community to each node
2. For each node i
    ▸ For each neighbor j of i, consider removing i from its
      community and placing it to j's community
    ▸ Greedily chose to place i into community of neighbor that
      leads to highest modularity gain
3. Repeat until no improvement can be done
```

No positive gain,
i stays in its original community

1. One node may be considered multiple times
2. Stops when local maxima is attained (no individual move can improve the Q)

Modularity gain $\Delta Q=Q(t)-Q(t-1)$

$$Q=\frac{1}{2m}\sum_{i,j}\left[A_{ij}-\frac{k_ik_j}{2m}\right]\delta(c_i,c_j),$$

Louvain algorithm

Pseudocode for phase 2

Building a new network

1. Let each community C_i form a new node i
2. Let the edges between new nodes i and j be the sum of edges between nodes in C_i and C_j in the previous graph (notice there are self-loops)

Links between nodes in same community leads to self loop

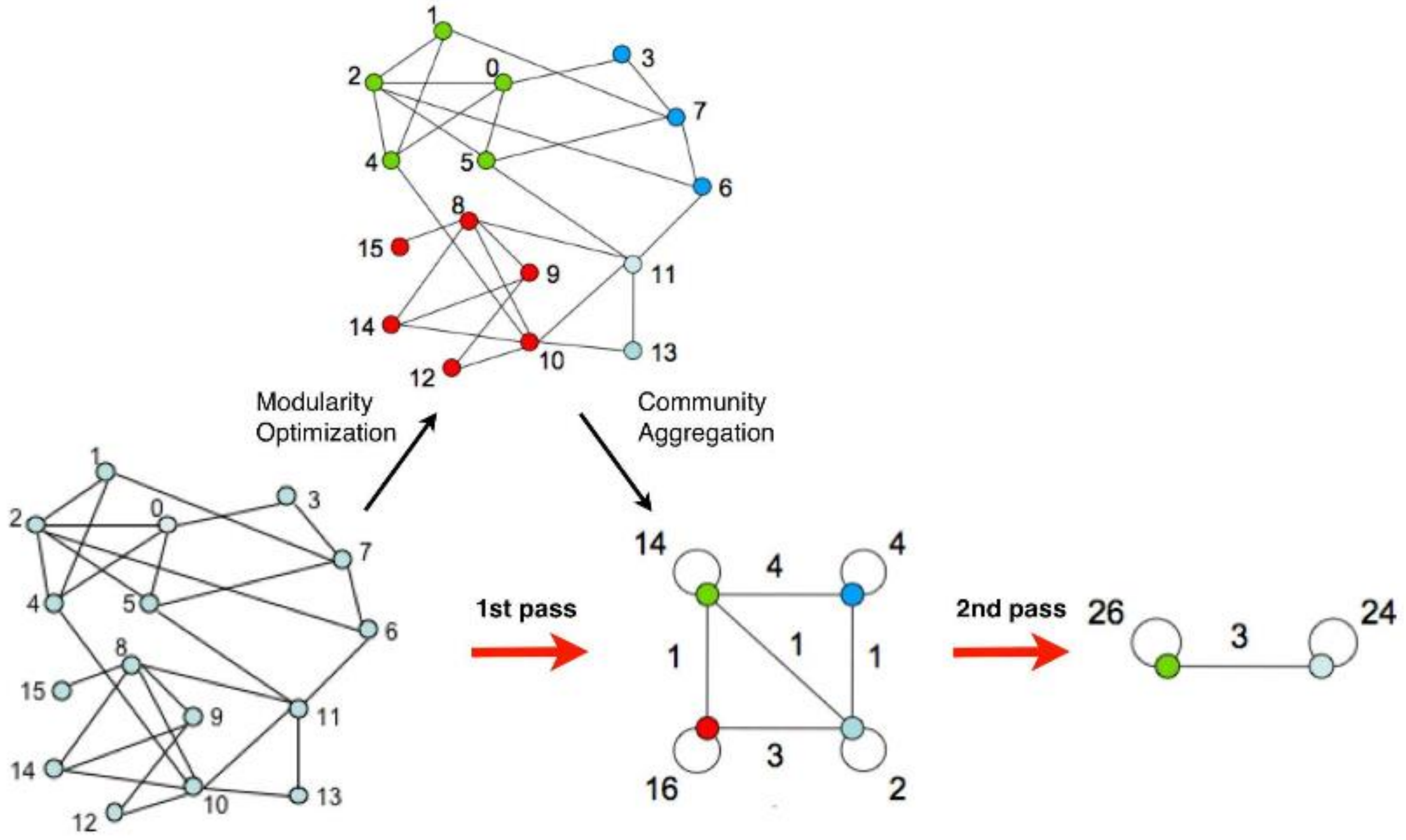
Once we complete the phase 2, reapply the phase 1 on the new graph

Pass2, Pass 3……until maximum Q is attained

Number of meta communities deceases at each pass ..

Most of the computing time is used in phase 1

Louvain algorithm



Communities of communities are built during the process
Height of the hierarchy is determined by the number of passes

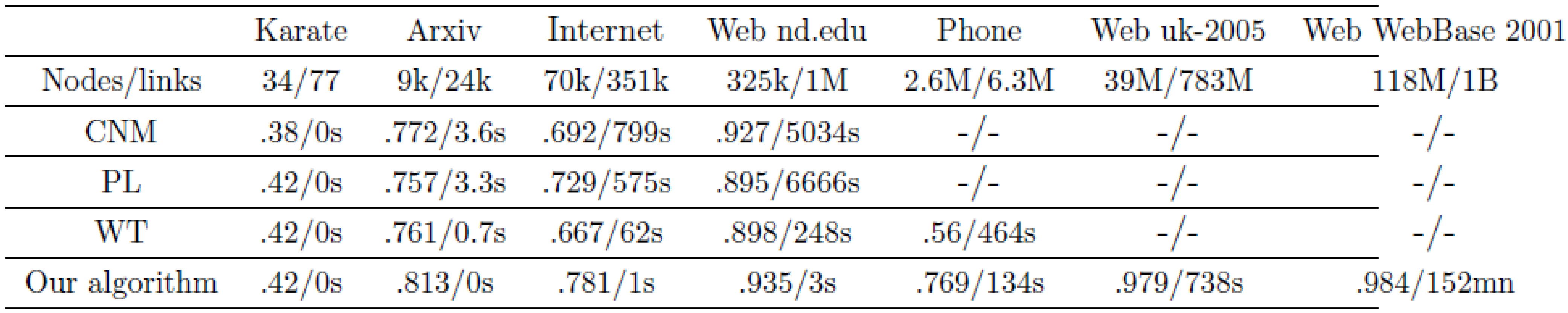
Louvain algorithm

- The output is also a hierarchy
- Works for weighted graphs, and so modularity has to be generalized to

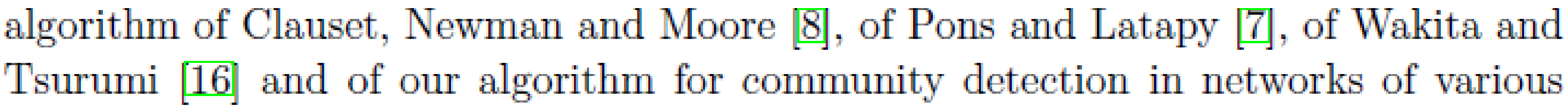
$$Q^w = \frac{1}{2W} \sum_{ij} \left(W_{ij} - \frac{s_i s_j}{2W} \right) \delta(C_i, C_j)$$

where W_{ij} is the weight of undirected edge (i, j),
$W = \sum_{ij} W_{ij}$ and $s_i = \sum_k W_{ik}$.

Louvain algorithm

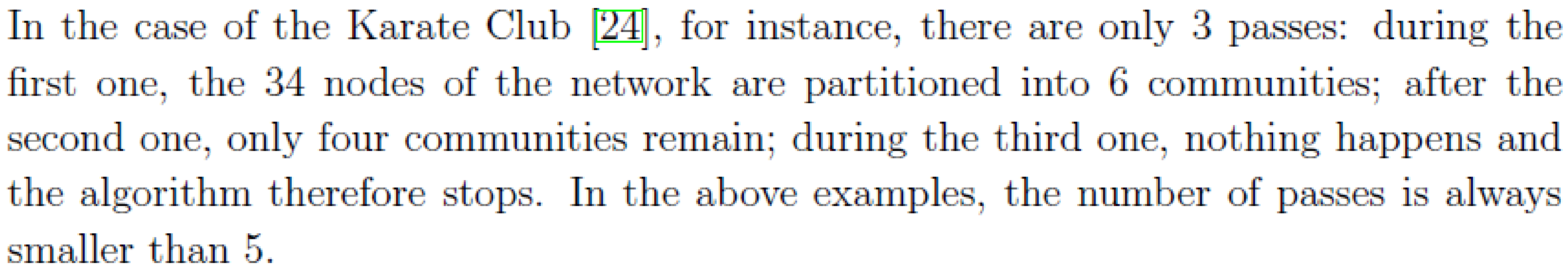


	Karate	Arxiv	Internet	Web nd.edu	Phone	Web uk-2005	Web WebBase 2001
Nodes/links	34/77	9k/24k	70k/351k	325k/1M	2.6M/6.3M	39M/783M	118M/1B
CNM	.38/0s	.772/3.6s	.692/799s	.927/5034s	-/-	-/-	-/-
PL	.42/0s	.757/3.3s	.729/575s	.895/6666s	-/-	-/-	-/-
WT	.42/0s	.761/0.7s	.667/62s	.898/248s	.56/464s	-/-	-/-
Our algorithm	.42/0s	.813/0s	.781/1s	.935/3s	.769/134s	.979/738s	.984/152mn



algorithm of Clauset, Newman and Moore [8], of Pons and Latapy [7], of Wakita and Tsurumi [16] and of our algorithm for community detection in networks of various

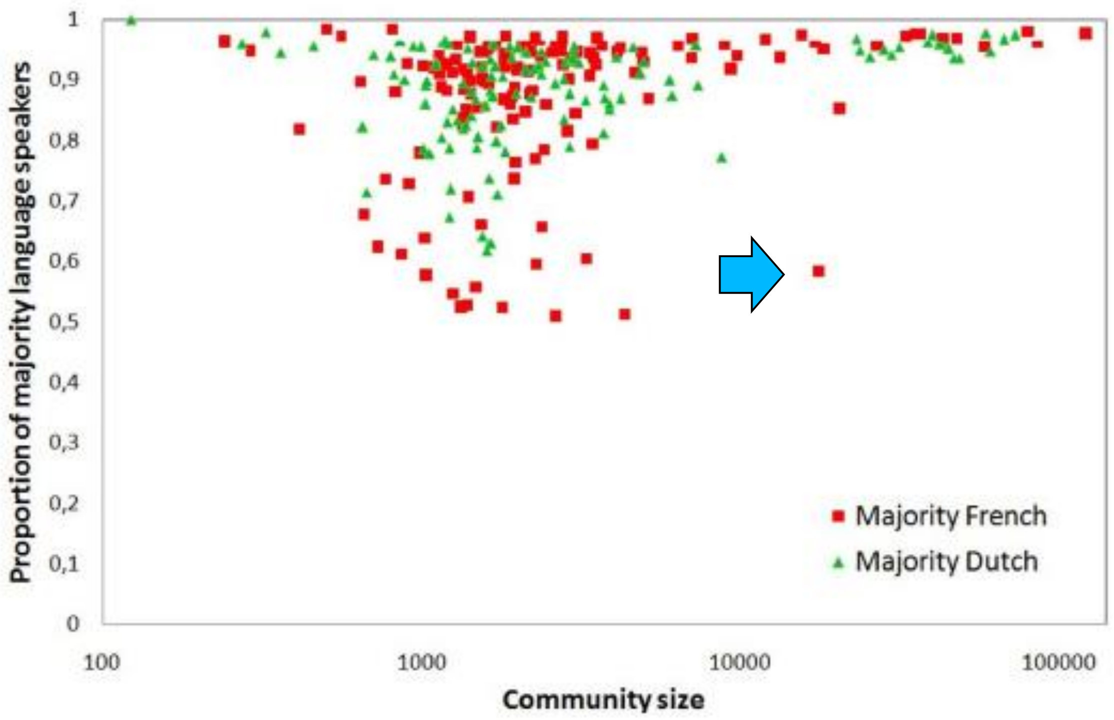
Fast and high modularity obtained



In the case of the Karate Club [24], for instance, there are only 3 passes: during the first one, the 34 nodes of the network are partitioned into 6 communities; after the second one, only four communities remain; during the third one, nothing happens and the algorithm therefore stops. In the above examples, the number of passes is always smaller than 5.

Belgian phone call network

2.6M customers, weighted links --- total phone calls during 6 months

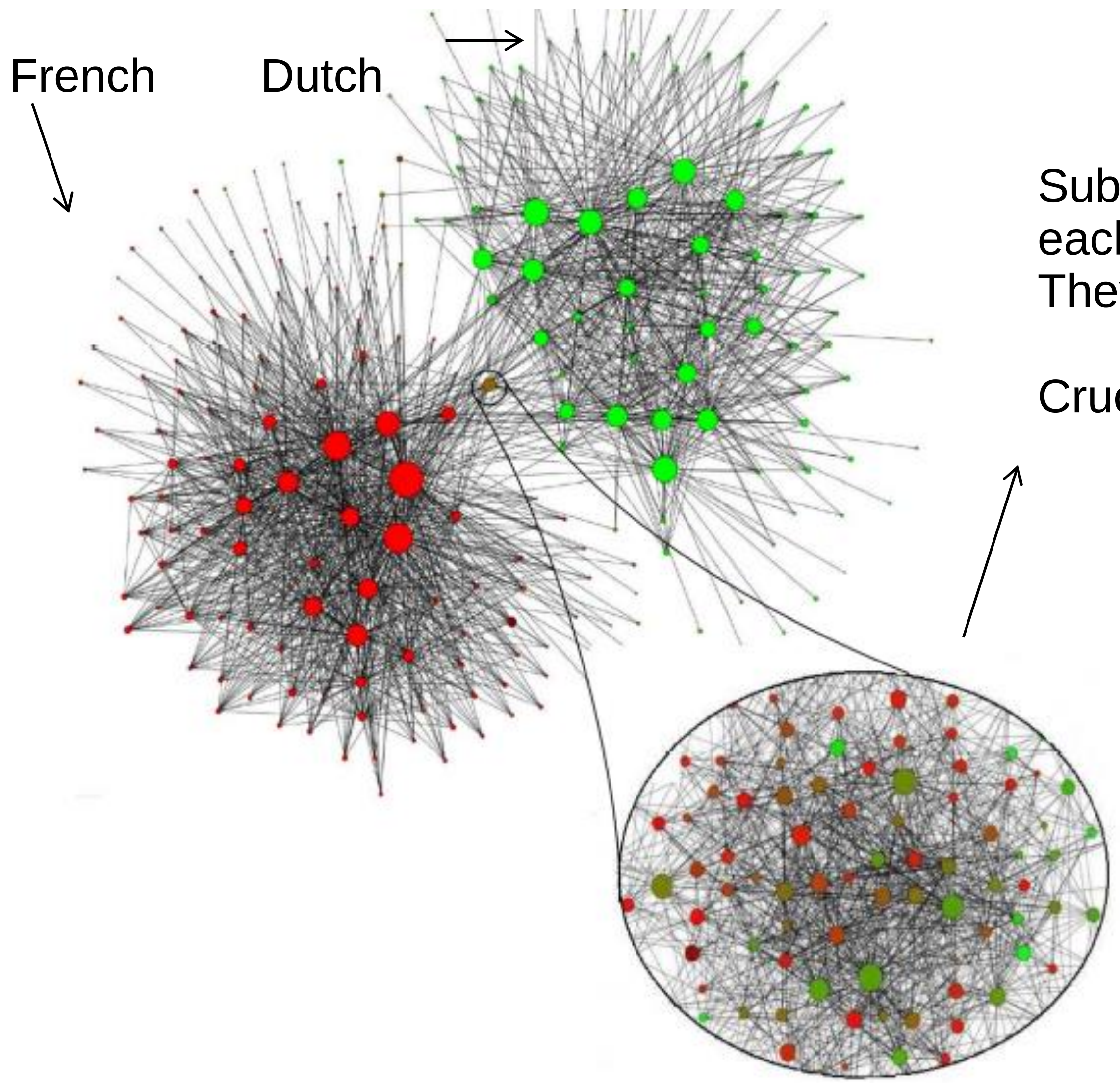


Six levels
Top level-------261 communities (>100 customers, >75% of total population)

Homogeneity of a community is measured by the fraction of people speaking in dominant language

Most of the communities are monolingual

36 communities of size>10000
Except **one,** all these communities have >85% speaking in one language



Sub communities are closely connected to each other
They also composed of heterogeneous groups

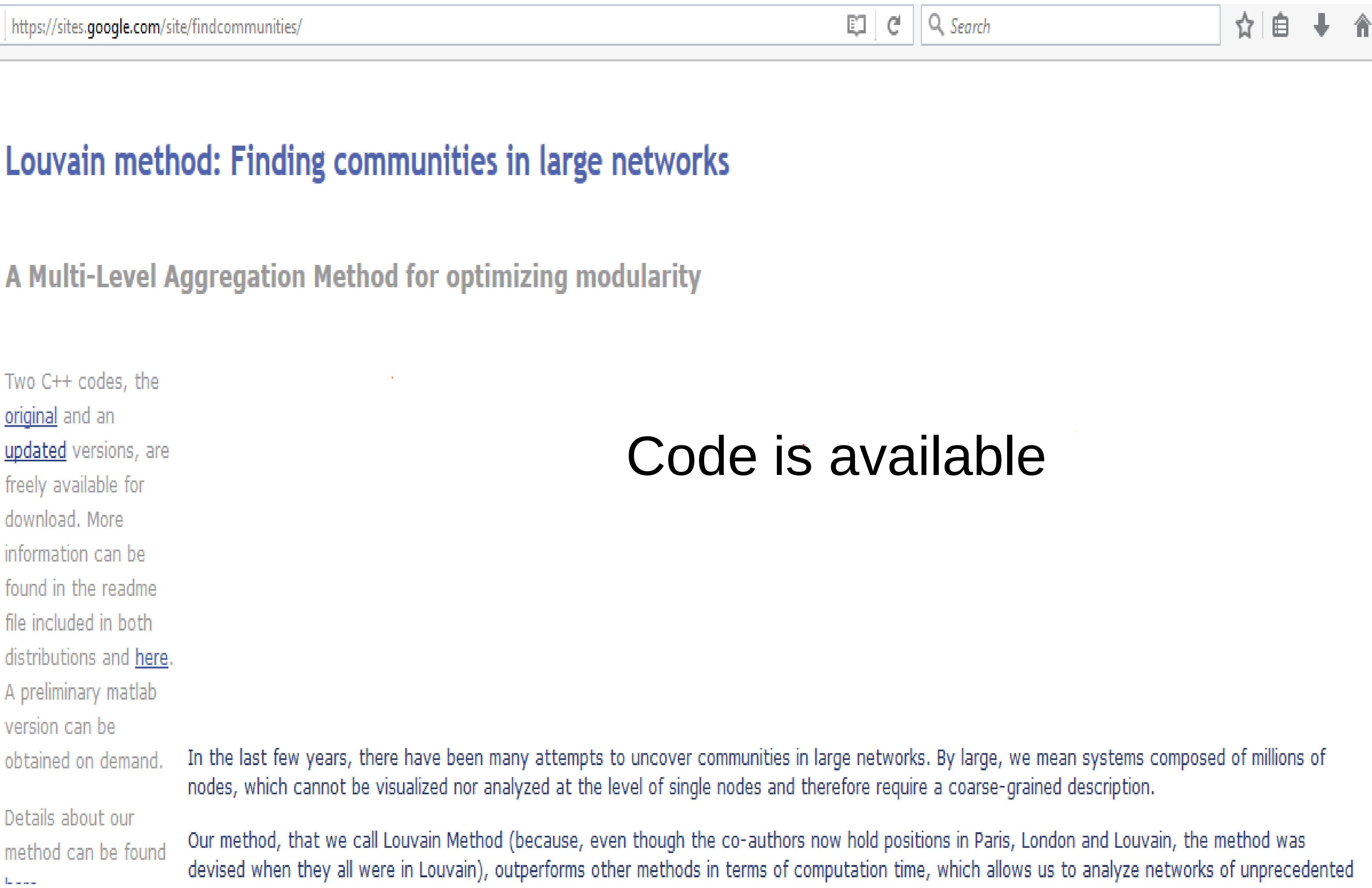
Crucial for integration of the country

Presence of German and English

66% German is one community

French people more densely connected

Different social behavior


https://sites.google.com/site/findcommunities/

Louvain method: Finding communities in large networks

A Multi-Level Aggregation Method for optimizing modularity

Two C++ codes, the original and an updated versions, are freely available for download. More information can be found in the readme file included in both distributions and here. A preliminary matlab version can be obtained on demand.

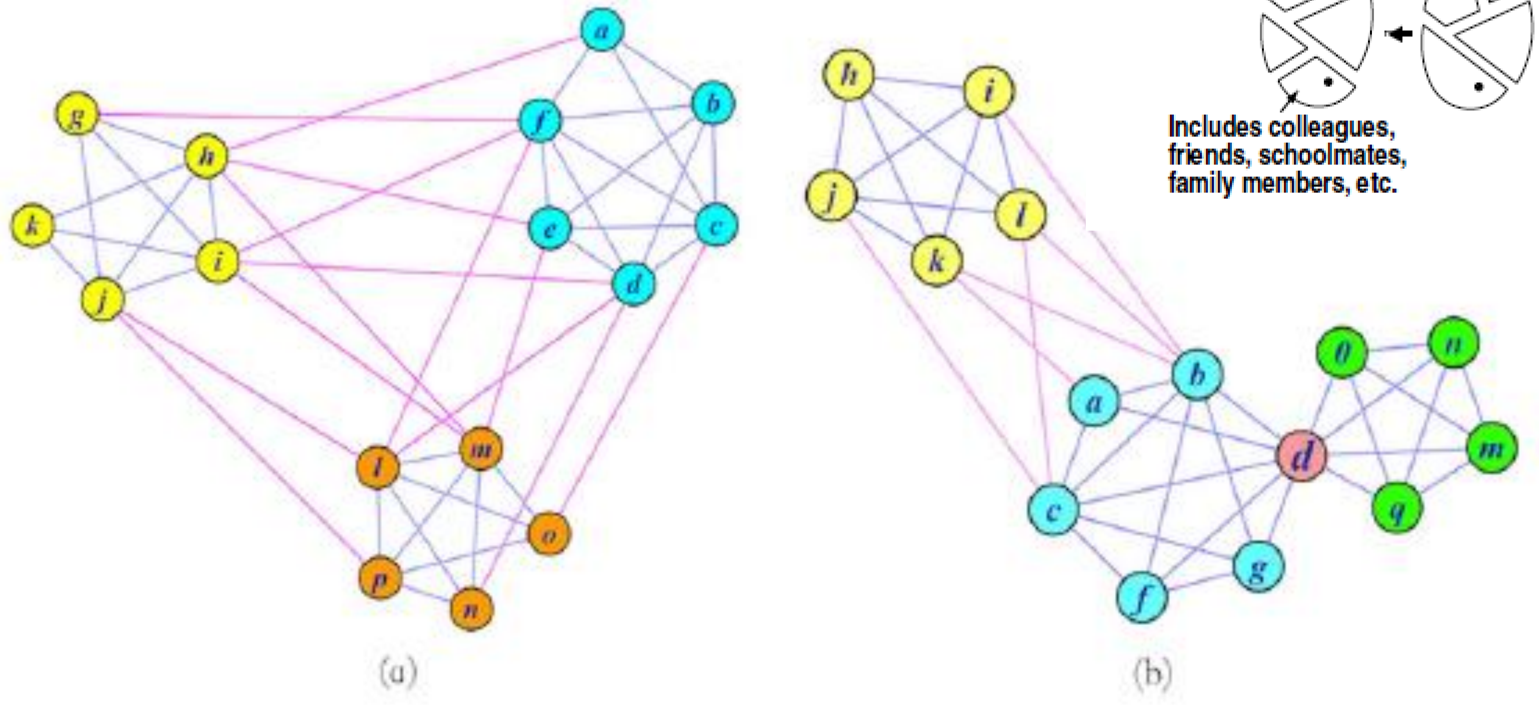
Details about our method can be found

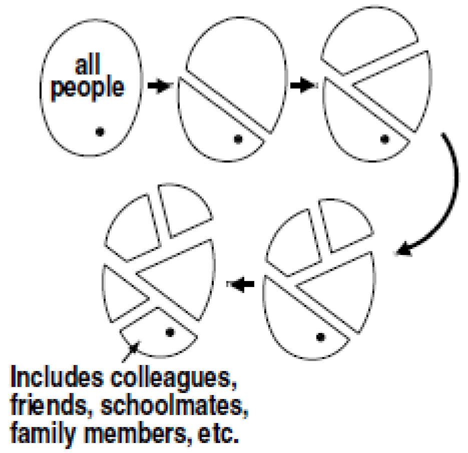
In the last few years, there have been many attempts to uncover communities in large networks. By large, we mean systems composed of millions of nodes, which cannot be visualized nor analyzed at the level of single nodes and therefore require a coarse-grained description.

Our method, that we call Louvain Method (because, even though the co-authors now hold positions in Paris, London and Louvain, the method was devised when they all were in Louvain), outperforms other methods in terms of computation time, which allows us to analyze networks of unprecedented

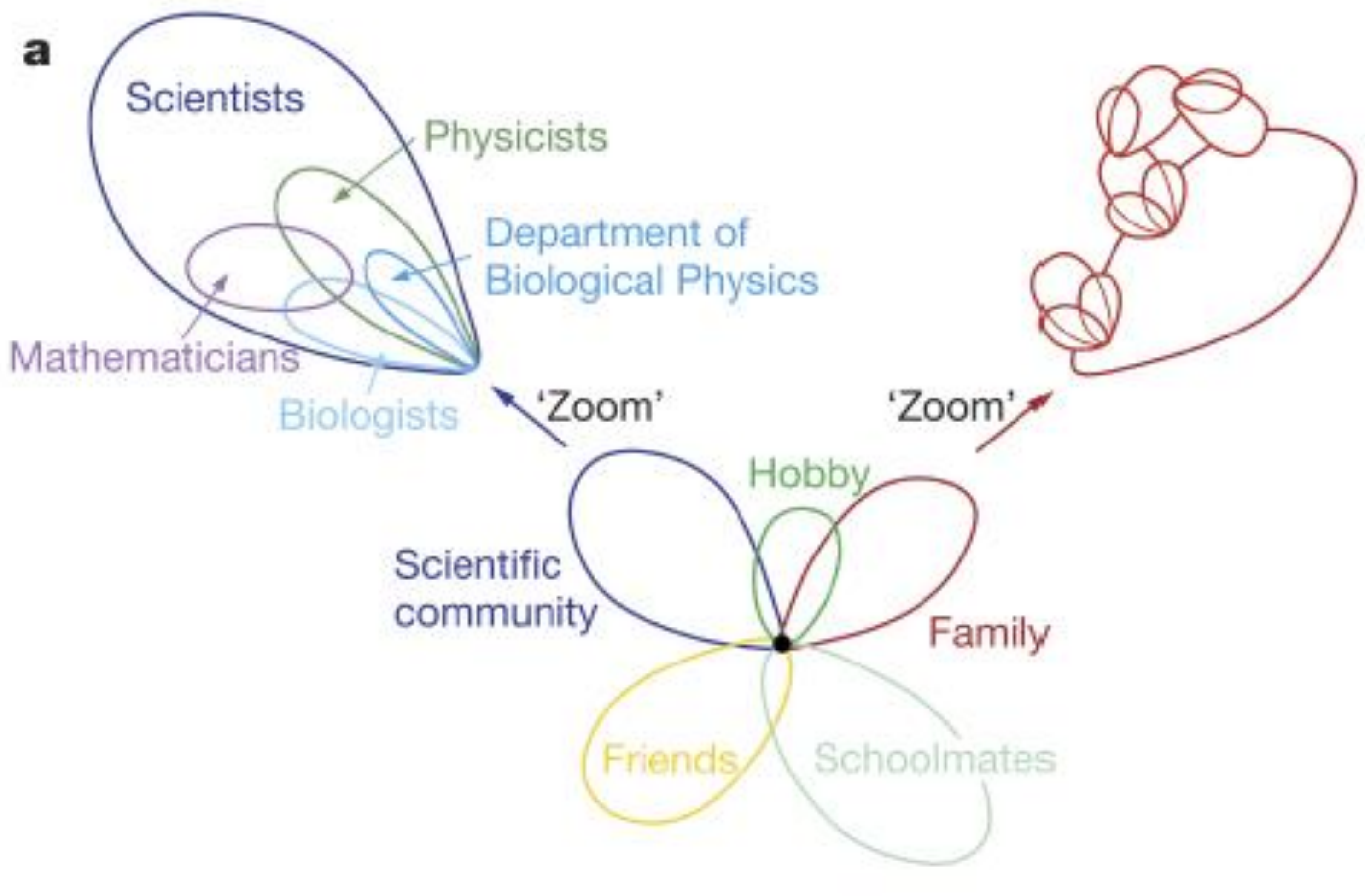
Code is available

Overlapping communities

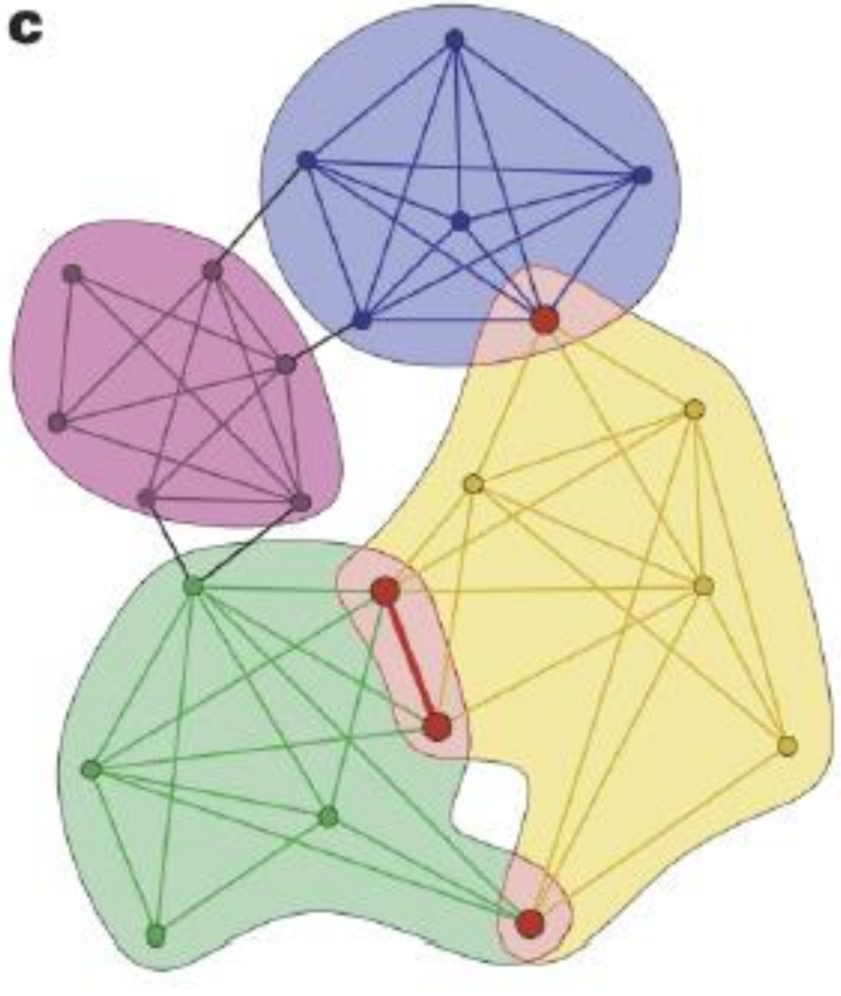



all people
Includes colleagues, friends, schoolmates, family members, etc.

Overlapping communities



Overlapping communities



Clique percolation

- Our community definition is based on the observation that a typical member in a community is linked to many other members, but not necessarily to all other nodes in the community
- In other words, a community can be interpreted as a union of smaller complete (fully connected) subgraphs that share nodes.
  - such complete subgraphs are called k-*cliques*, where k refers to the number of nodes in the subgraph

k-clique template rolling

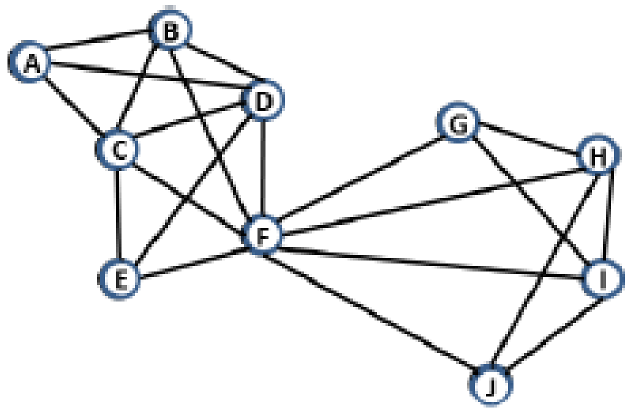
k-clique template: A k-clique template can be thought of as an object that is isomorphic to a complete graph of k nodes.

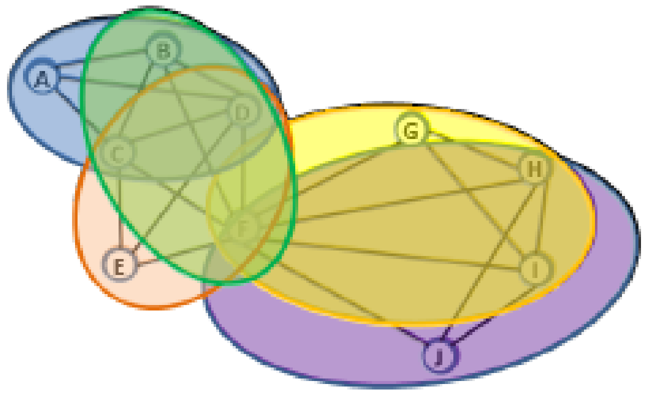
Such a template can be placed onto any k-clique of the network, and rolled to an adjacent k-clique by relocating one of its nodes and keeping its other k-1 nodes fixed.

Thus, the k-clique-communities of a graph are all those subgraphs that can be fully explored by rolling a k-clique template in them but cannot be left by this template.

K-clique community

- k-clique is a clique (complete subgraph) with k nodes
- k-clique community a union of all k-cliques that can be reached from each other through a series of adjacent k-cliques
- two k-cliques are said to be adjacent if they share $k-1$ nodes.

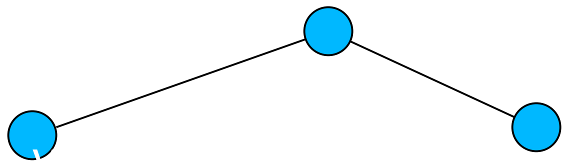




Adjacent 4-cliques

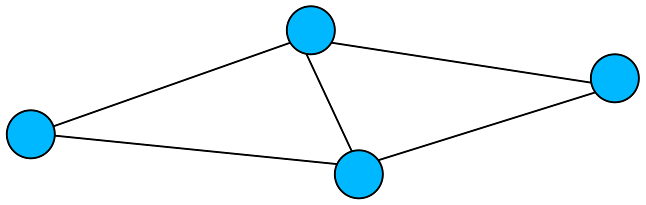
Special cases

The k-clique-communities of a network at k = 2 are equivalent to the connected components



K=2

Series of shared nodes



K=3

Series of shared edges

K-clique community

- k-*clique chain:* the union of a sequence of adjacent k-cliques
- k-*clique connectedness*: two k-cliques are k-clique-connected if they are parts of a k-clique chain.
- k-clique-communities are equivalent to the k-clique connected components of the network.

k-clique template rolling

k-clique template: A k-clique template can be thought of as an object that is isomorphic to a complete graph of k nodes.

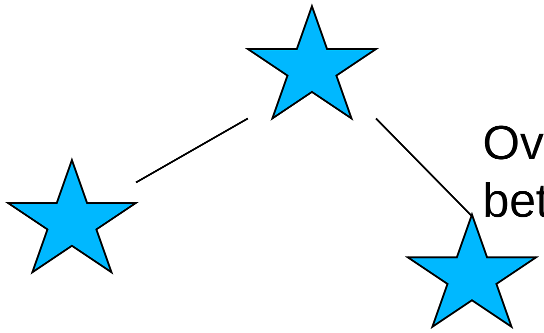
Such a template can be placed onto any k-clique of the network, and rolled to an adjacent k-clique by relocating one of its nodes and keeping its other k-1 nodes fixed.

Thus, the k-clique-communities of a graph are all those subgraphs that can be fully explored by rolling a k-clique template in them but cannot be left by this template.

K-clique percolation

first extracts all complete subgraphs of the network that are not parts of larger complete subgraphs

- Find all maximal cliques
- Create clique overlap matrix
- Threshold matrix at value $k-1$
- Communities = connected components



In this symmetric matrix each row (and column) represents a clique

the matrix elements are equal to the number of common nodes between the corresponding two cliques

(Note that the intersection of two cliques is always a complete subgraph.)

Diagonal entries are equal to the size of the clique.

K-clique percolation

These components can be found by erasing (i) every off-diagonal entry smaller than k-1 and (ii) every diagonal element smaller than k in the matrix,

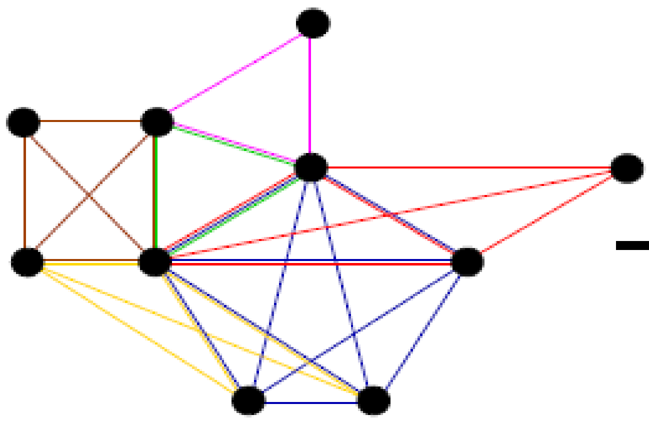
Replacing the remaining elements by 1

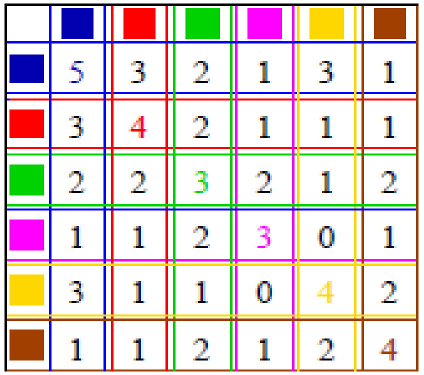
Carrying out a component analysis of this matrix

The k-clique-communities for a given value of k are equivalent to

such **connected** clique components in which the neighbouring cliques are linked to each other by at least k-1 common nodes.

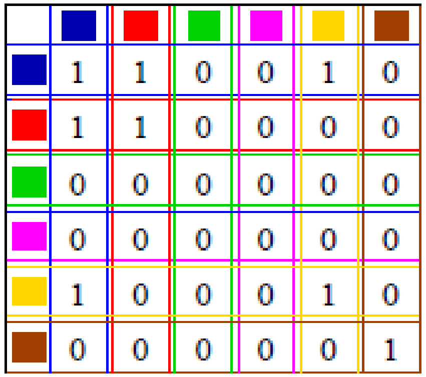
K-clique percolation



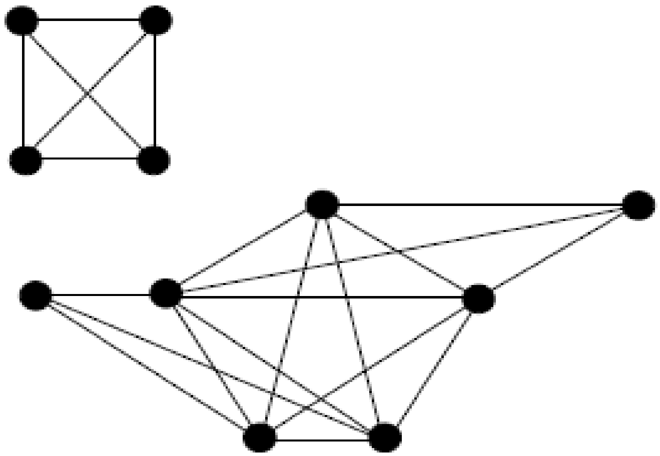


	🟦	🟥	🟩	🟪	🟨	🟫
🟦	5	3	2	1	3	1
🟥	3	4	2	1	1	1
🟩	2	2	3	2	1	2
🟪	1	1	2	3	0	1
🟨	3	1	1	0	4	2
🟫	1	1	2	1	2	4

k=4



	🟦	🟥	🟩	🟪	🟨	🟫
🟦	1	1	0	0	1	0
🟥	1	1	0	0	0	0
🟩	0	0	0	0	0	0
🟪	0	0	0	0	0	0
🟨	1	0	0	0	1	0
🟫	0	0	0	0	0	1



Mixing pattern

Network mixing patterns

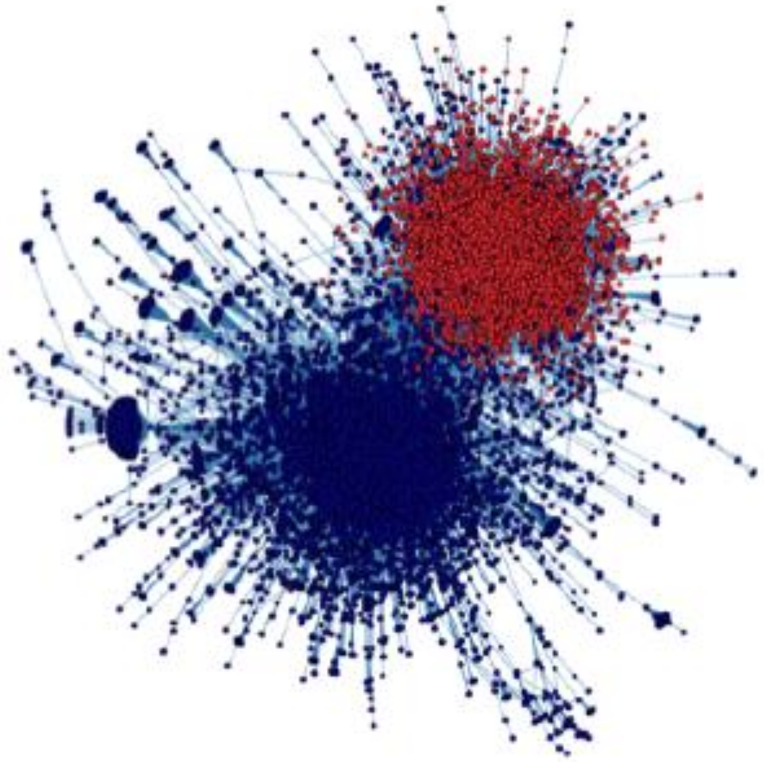
- **Assortative mixing**, "like links with like", attributed of connected nodes tend to be more similar than if there were no such edge
- **Disassortative mixing**, "like links with dislike", attributed of connected nodes tend to be less similar than if there were no such edge

Vertices can mix on any vertex attributes (age, sex, geography in social networks), unobserved attributes, vertex degrees

Examples:
assortative mixing - in social networks political beliefs, obesity, race
disassortative mixing - dating network, food web (predator/prey), economic networks (producers/consumers)

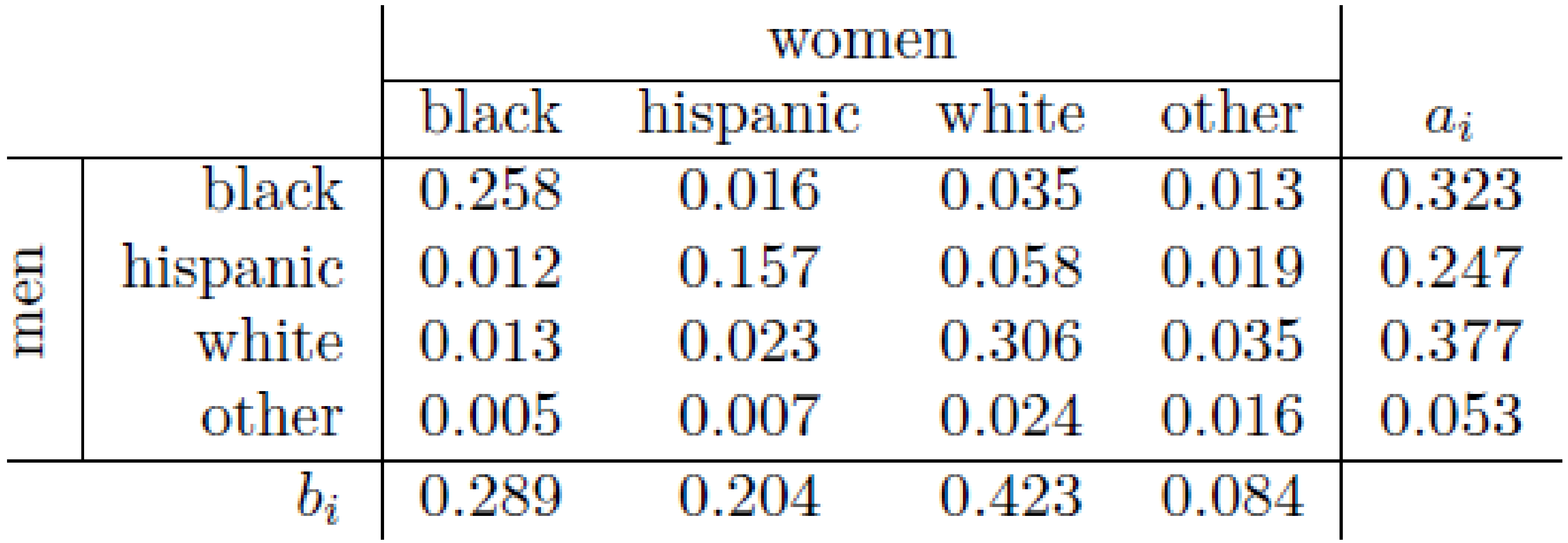
Mixing pattern

- Political polarization on Twitter: political retweet network ,red color - "right-learning" users, blue color - "left learning" users



- Assortative mixing = homophily

Assortative mixing



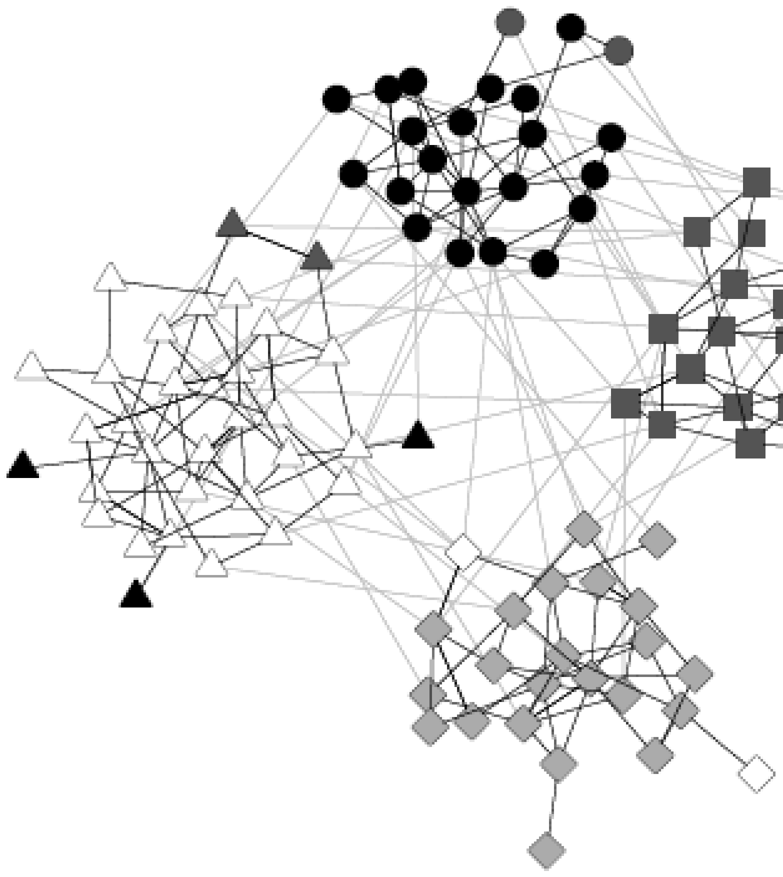
		women				
		black	hispanic	white	other	a_i
men	black	0.258	0.016	0.035	0.013	0.323
	hispanic	0.012	0.157	0.058	0.019	0.247
	white	0.013	0.023	0.306	0.035	0.377
	other	0.005	0.007	0.024	0.016	0.053
	b_i	0.289	0.204	0.423	0.084	

The amount of assortative mixing in a network can be characterized by measuring how much of the weight in the mixing matrix falls on the diagonal, and how much off it.

Let us define e_{ij} to be the fraction of all edges in a network that join a vertex of type i to a vertex of type j.

Ends of an edge always attach to one man and one woman, we also specify which index corresponds to which type of end, which makes **e** asymmetric

Friendship network



$$e = \begin{pmatrix} 0.18 & 0.02 & 0.01 & 0.03 \\ 0.02 & 0.20 & 0.03 & 0.02 \\ 0.01 & 0.03 & 0.16 & 0.01 \\ 0.03 & 0.02 & 0.01 & 0.22 \end{pmatrix}^{a_i}_{b_i},$$

Assortative mixing

The matrix should also satisfy the sum rules

$$\sum_{ij} e_{ij} = 1, \qquad \sum_{j} e_{ij} = a_i, \qquad \sum_{i} e_{ij} = b_j,$$

where a_i and b_i are the fraction of each type of end of an edge that is attached to vertices of type i

$$r = \frac{\sum_i e_{ii} - \sum_i a_i b_i}{1 - \sum_i a_i b_i}$$

the sum of the elements of the matrix **x**. We call the quantity r the "assortativity coefficient". It takes the value 1 in a perfectly assortative network, since in that case the entire weight of the matrix **e** lies along its diagonal and $\sum_i e_{ii} = 1$. Conversely, if there is no assortative mixing at all, then $e_{ij} = a_i b_j$ for all i, j and $r = 0$. Networks can also be disassortative: vertices may associate preferentially with others of different types—the "opposites attract" phenomenon. In that case, r will take a negative value.

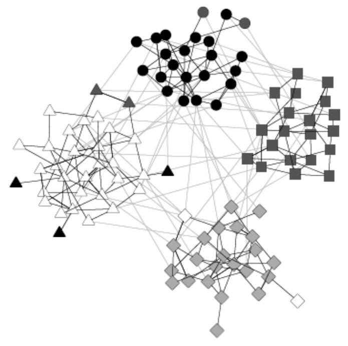
Assortativity leads to community formation

Degree distribution p(k)

$$e = \begin{pmatrix} 0.18 & 0.02 & 0.01 & 0.03 \\ 0.02 & 0.20 & 0.03 & 0.02 \\ 0.01 & 0.03 & 0.16 & 0.01 \\ 0.03 & 0.02 & 0.01 & 0.22 \end{pmatrix},$$

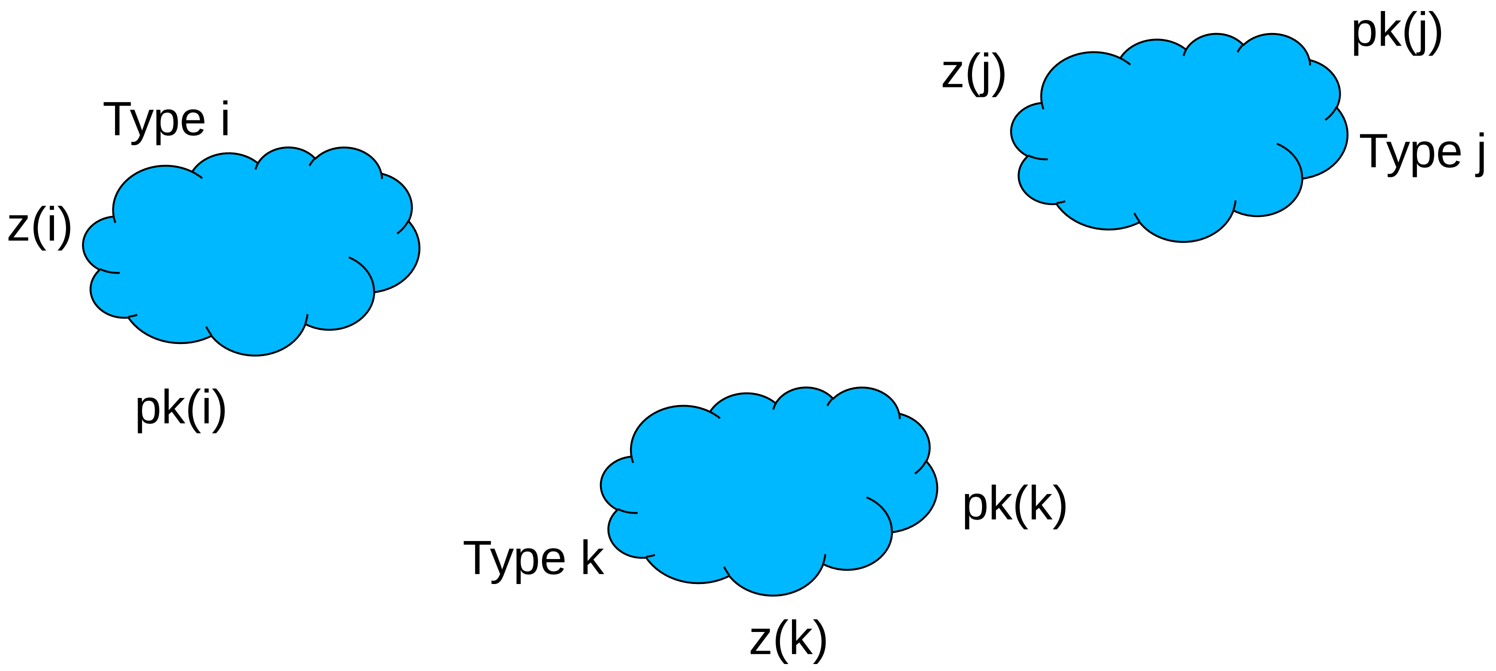
$$\boxed{\text{Model}}$$

Graph



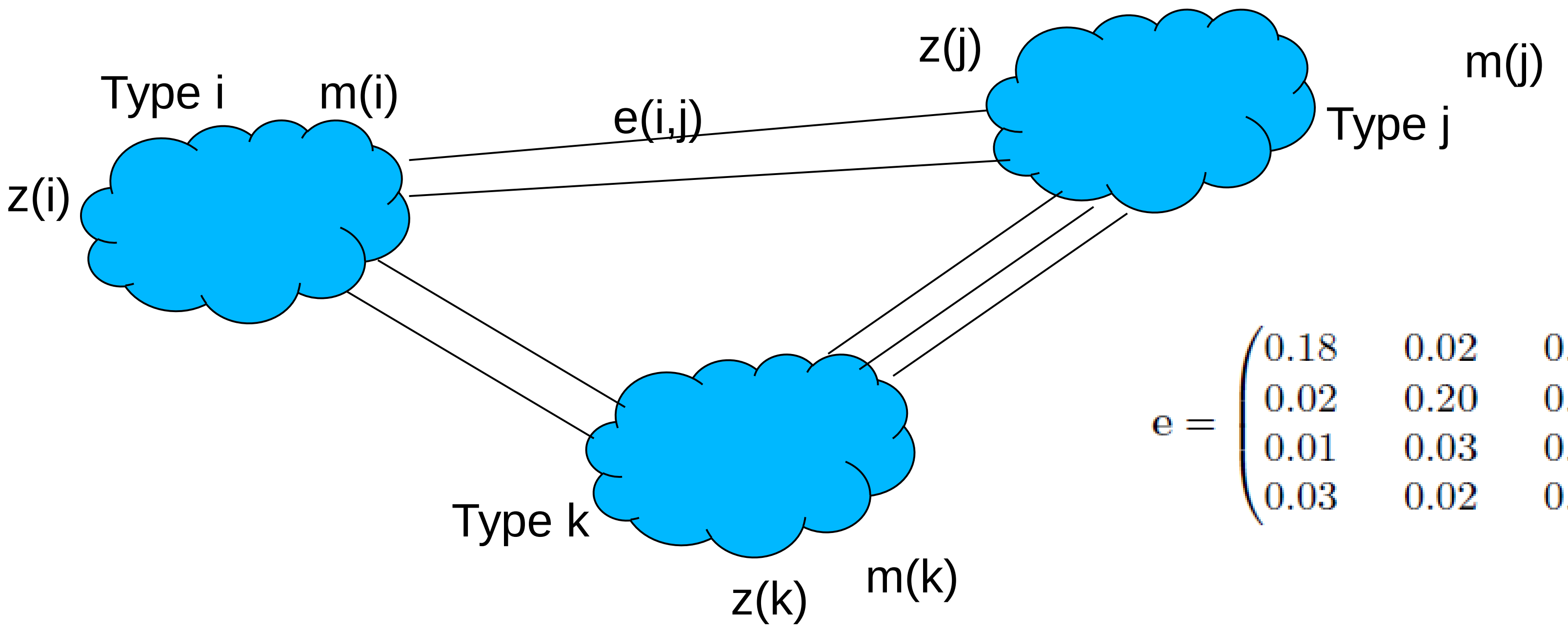
Apply community detection algo

1. First we choose degree distributions $p_k^{(i)}$ for each vertex type i. The quantity $p_k^{(i)}$ here denotes the probability that a randomly chosen vertex of type i will have degree k. We can also calculate the mean degree $z_i = \sum_k k p_k^{(i)}$ for each vertex type.



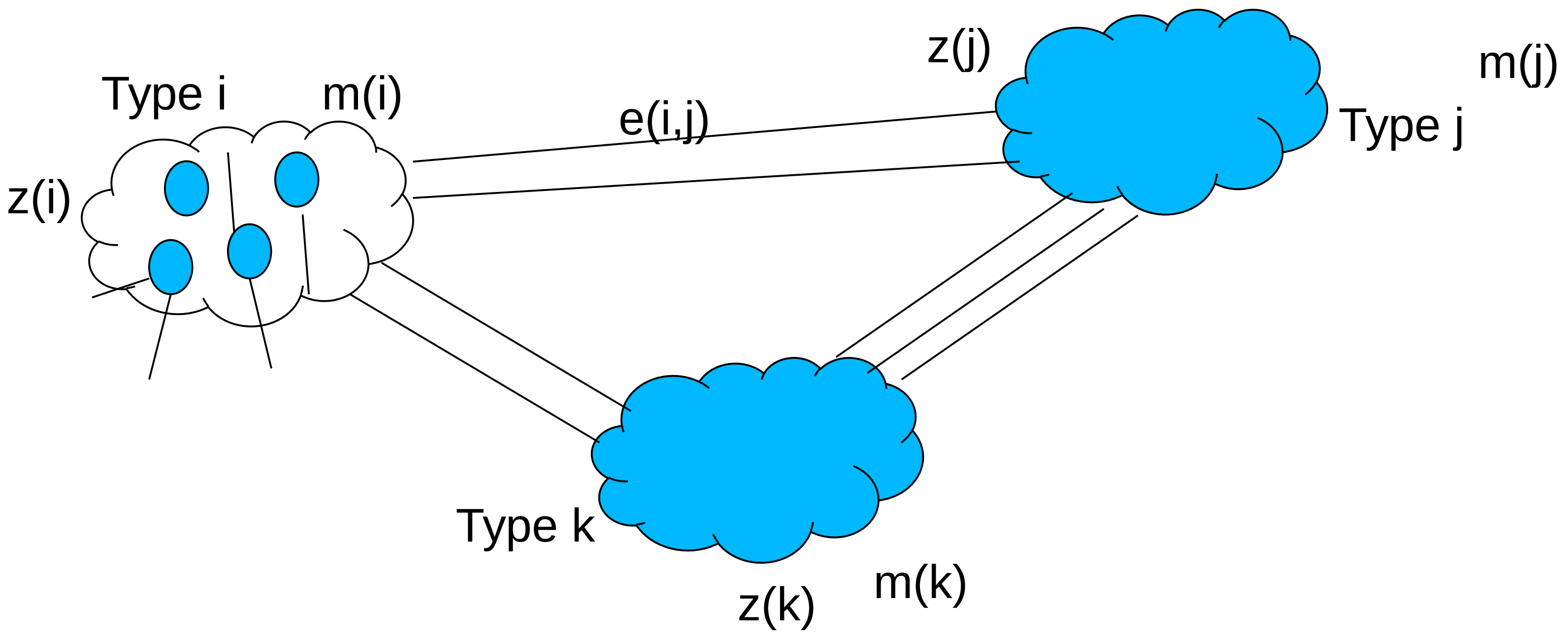
2. Next we choose a size for our graph in terms of the number m of edges and draw m edges from the desired distribution e_{ij}. We count the number of ends of edges of each type i, to give the sums m_i of the degrees of vertices in each class, and we calculate the expected number n_i of vertices of each type from $n_i = m_i/z_i$ (rounded to the nearest integer).

Compute m(i), m(j) etc



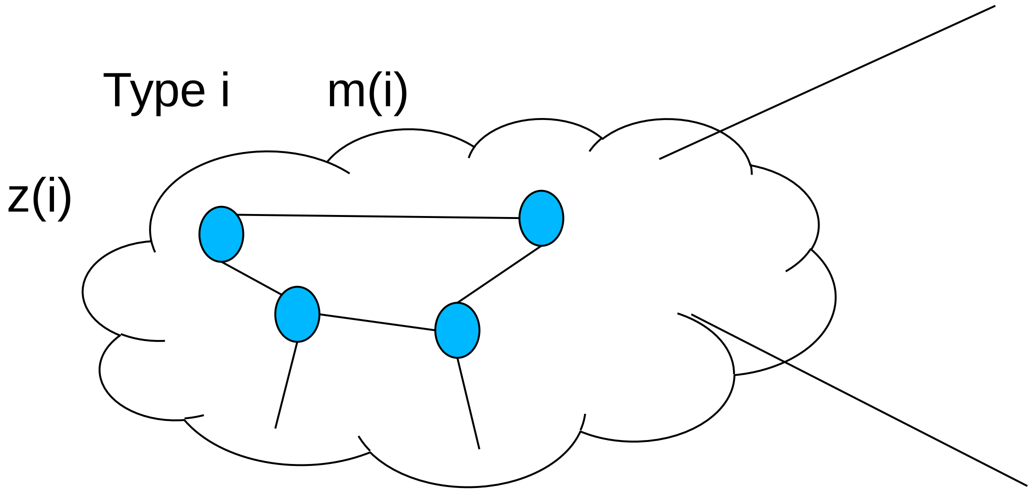
$$e = \begin{pmatrix} 0.18 & 0.02 & 0.01 & 0.03 \\ 0.02 & 0.20 & 0.03 & 0.02 \\ 0.01 & 0.03 & 0.16 & 0.01 \\ 0.03 & 0.02 & 0.01 & 0.22 \end{pmatrix},$$

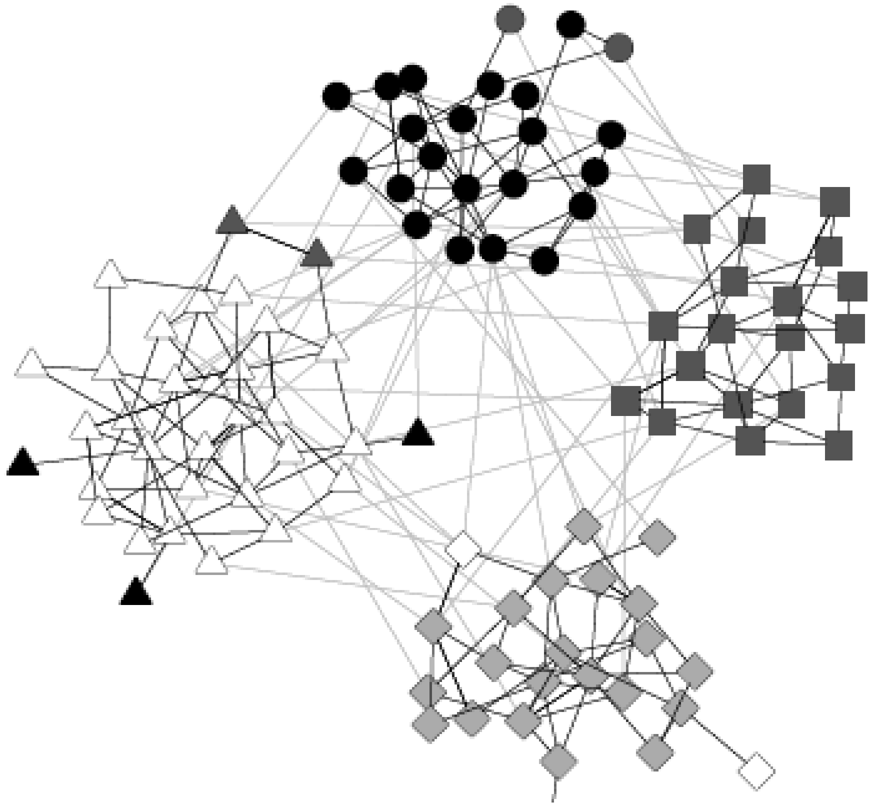
3. We draw n_i vertices from the desired degree distribution $p_k^{(i)}$ for type i. Normally the degrees of these vertices will not sum exactly to m_i as we want them to, in which case we choose one vertex at random, discard it, and draw another from the distribution $p_k^{(i)}$, repeating until the sum does equal m_i.



4. We pair up the m_i ends of edges of type i at random with the vertices we have generated, so that each vertex has the number of attached edges corresponding to its chosen degree.

5. We repeat from step 3 for each vertex type.





$$e = \begin{pmatrix} 0.18 & 0.02 & 0.01 & 0.03 \\ 0.02 & 0.20 & 0.03 & 0.02 \\ 0.01 & 0.03 & 0.16 & 0.01 \\ 0.03 & 0.02 & 0.01 & 0.22 \end{pmatrix},$$

Figure 6: A network generated using the mixing matrix of Eq. (3) and a Poisson degree distribution with mean $z = 5$. The four different shades of vertices represent the four types, and the four shapes represent the communities discovered by the community-finding algorithm of Section 2.1. The placement of the vertices has also been chosen to accentuate the communites and show where the algorithm fails. As we can see, the correspondence between vertex type and the detected community structure is very close; only nine of the 100 vertices are misclassified.