# Lecture 5: Graph Partitioning and Community Detection

# *Foreword*

***Computational Complexity:*** A measure of the running time of a computer algorithm.

Simple examples from Python coding:

```
def ex3a(n):
    print("hi")          # 1
    print("hi")          # 1 ;
                         # 1 + 1 = 2 ~ O(1)

def ex3b(n):
    k = 0                # 1
    while (k < 10):      # 10 + 1
        print("hi")      # 10
        k = k + 1        # 10
                         # 1 + 10 + 1 + 10 + 10 = 32 ~ O(1)

def ex3c(n):
    k = -10              # 1
    while (k <= 10):     # 21 + 1
        print("hi")      # 21
        k = k + 1        # 21
                         # 1 + 21 + 1 + 21 + 21 = 65 ~ O(1)

def ex3d(n):
    while (n > 1):       # n
        print("hi")      # n - 1
        n = n - 1        # n - 1
                         # n + n + n - 1 - 1 = 3*n - 2 ~ O(n)

def ex3e(n):
    k = n                # 1
    while (k >= -n):     # 2*n + 1 + 1
        print("hi")      # 2*n + 1
        k = k - 1        # 2*n + 1
                         # 3*(2*n + 1) + 1 + 1 = 6*n + 5 ~ O(n)

def ex3f(n):
    k = -5               # 1
    while (k <= 2*n):    # 2*n + 6 + 1
        print("hi")      # 2*n + 6
        k = k + 1        # 2*n + 6
                         # 3*(2*n + 6) + 2 = 6*n + 20 ~ O(n)
```

```
def ex3g(n):
    i = 1                          # 1
    while (i <= n):                # n + 1
        j = 1                      # n
        while (j <= n):            # n*(n + 1)
            print("hi")            # n*n
            j = j + 1              # n*n
        i = i + 1                  # n
                                   # 3*n*n + 4*n + 2 ~ O(n^2)

def ex3h(n):
    i = 1                          # 1
    while (i <= n):                # n + 1
        j = 1                      # n
        while (j <= i):            # n*(n + 1)/2 + n
            print("hi")            # n*(n + 1)/2
            j = j + 1              # n*(n + 1)/2
        i = i + 1                  # n
                                   # 3*n*(n + 1)/2 + 4*n + 2 ~ O(n^2)
```

*****How many steps does it take to compute the triangle number?***

tri(5) = 1 + 2 + 3 + 4 + 5 = 15

```
def tri(n):                 # tri(5):      tri(n):
    total = 0               # 1            1
    while (n > 0):          # 5 * 3        3 * n
        total = total + n   #
        n = n - 1           #
    return(total)           # 1            1
                            # Total: 17    Total: 3*n + 2 ~ 3*n

def triangle(n):
    total = int( n * (n + 1) / 2 )
    return(total)                          # Total: 2 steps
```

# *Graph Partitioning and Community Detection*

There are a number of reasons why one might want to divide a network into groups or clusters, and they separate into two general classes that lead in turn to two corresponding types of computer algorithm, namely ***graph partitioning*** and ***community detection*** algorithms.

***Graph partitioning*** is a classic problem in computer science since 1960s, and is the problem of dividing the vertices of a network into a given number of non-overlapping groups of given sizes such that the number of edges between groups is minimized. The *important point* is that the number and sizes of the groups are fixed (sometimes vaguely fixed within a certain range).

***Community detection*** differ from graph partitioning in that the *number* and *size* of the groups into which the network divided are not specified. One wants to find the natural division among the nodes in clusters.

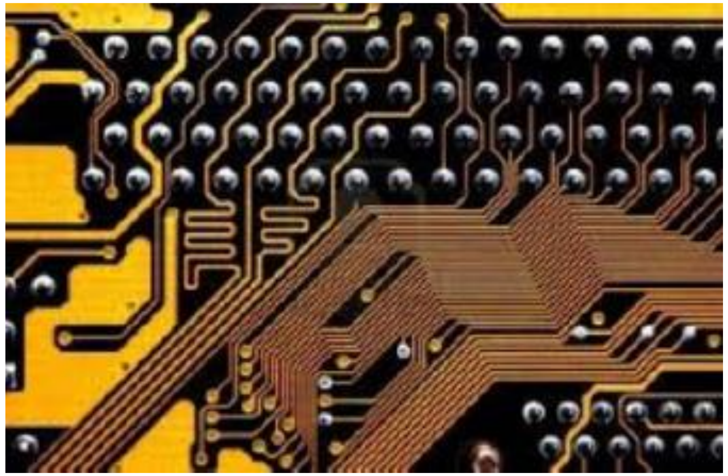
## *Difference in Goals:*

***Graph partitioning*** is a way of dividing up a network into smaller more manageable pieces, for example to perform numerical calculations. ***Community detection*** is often used as a tool for understanding the structure of a network that may not be easily visible in the raw network topology.

# *Graph Partitioning*

## *Example:*

Partition the full wiring diagram of an integrated circuit into smaller subgraphs, so that they minimize the number of connections between them.



2.5 billion transistors

# *Graph Partitioning*

## *Problem Formulation:*

- Input: A weighted graph G = ($V$, $E$) with
  - Vertex set $V$. ($|V| = 2n$)
  - Edge Set $E$. ($|E| = e$)
  - Cost $c_{AB}$ for each edge ($A$, $B$) in $E$.
- Output: 2 subsets $X$ & $Y$ such that
  - $V = X \cup Y$ and $X \cap Y = \{\ \}$ (i.e. partition)
  - Each subset (group) has $n$ vertices
  - Total cost of edges "crossing" the partition is minimized.

- This problem is NP-Complete !!!!!

# Graph Partitioning

## *Brute Force Method:*

- Try ***all*** possible bisections. Choose the best one.
- If there are $2n$ vertices
  - # of possibilities = $(2n)! / (n!)^2$
- For 4 vertices (A,B,C,D), 3 possibilities
  1. $X = \{A, B\}$ & $Y = \{C, D\}$
  2. $X = \{A, C\}$ & $Y = \{B, D\}$
  3. $X = \{A, D\}$ & $Y = \{B, C\}$
- For 100 vertices,
  - $5 \times 10^{28}$ possibilities!
- Brute force not possible!

# *Graph Partitioning*

## *Graph Partitioning Algorithms:*

Two well-known methods for graph partitioning.

- ***Kernighan-Lin algorithm*** (not based on matrix methods but provides a simple introduction to the partitioning Problem)

- ***Spectral Partitioning method*** (based on the spectral properties of the graph Laplacian)

# Graph Partitioning

## *Kernighan-Lin Algorithm:*

"*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 $10^{20}$.

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

# Graph Partitioning

## *Kernighan-Lin Algorithm:*

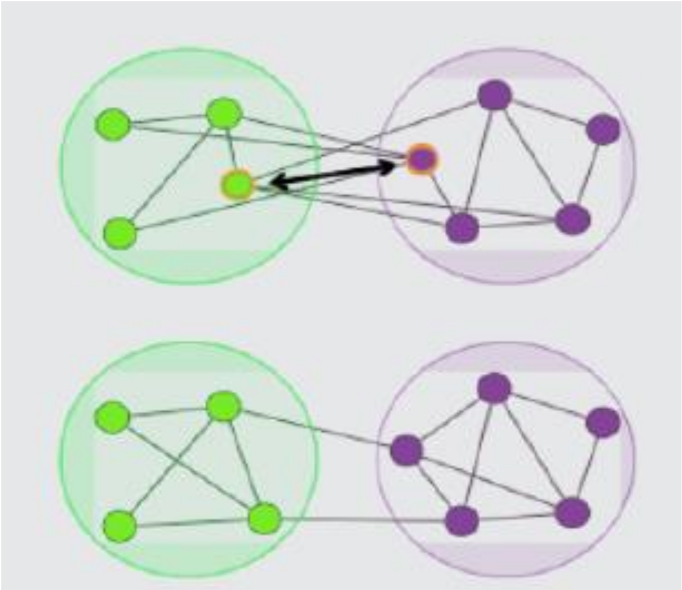
- Partition a network into two groups of predefined size. This partition is called ***cut***.

- Inspect each pair of nodes, one from each group. Identify the pair that results in the largest reduction of the ***cut size*** (links between the two groups) if we swap them.

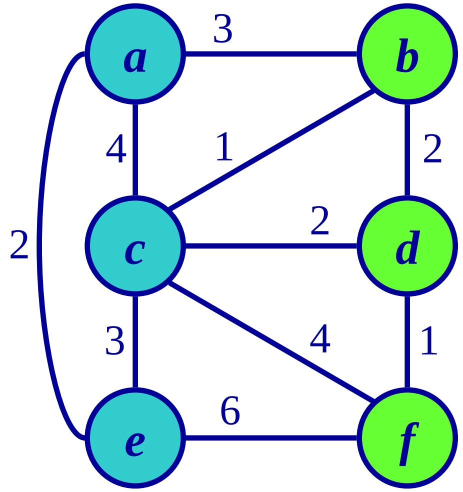
- Swap them.

- If no pair reduces the cut size, we swap the pair that increases the cut size the least.

- The process is repeated until each node is moved once.



# *Kernighan-Lin Algorithm:*




## ***Example:***

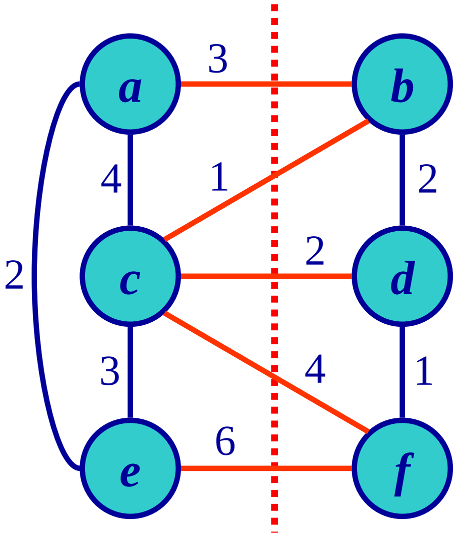
Given:

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

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

# *Kernighan-Lin Algorithm:*




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:

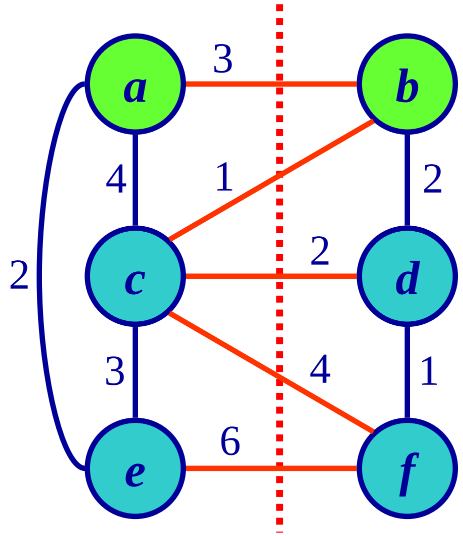
$G_x = E_x - I_x$

$E_x$ = cost of edges connecting node x with the other group (extra)

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

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

# Kernighan-Lin Algorithm:




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

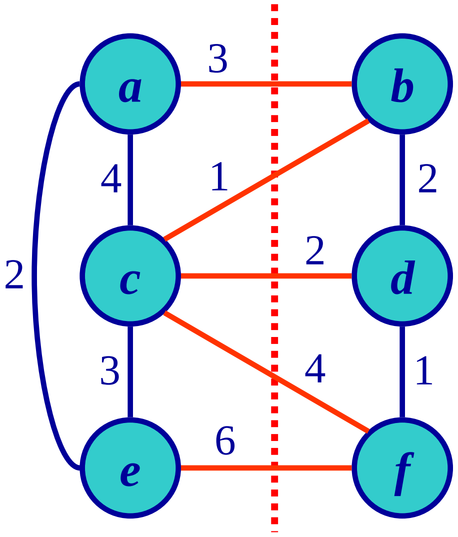
However, the cost saving 3 of the direct edge was counted twice. But this edge still connects the two groups

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

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

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

# Kernighan-Lin Algorithm:




cut-size = 16

Compute all the gains

$G_a = -3$ $\quad G_b = +2$

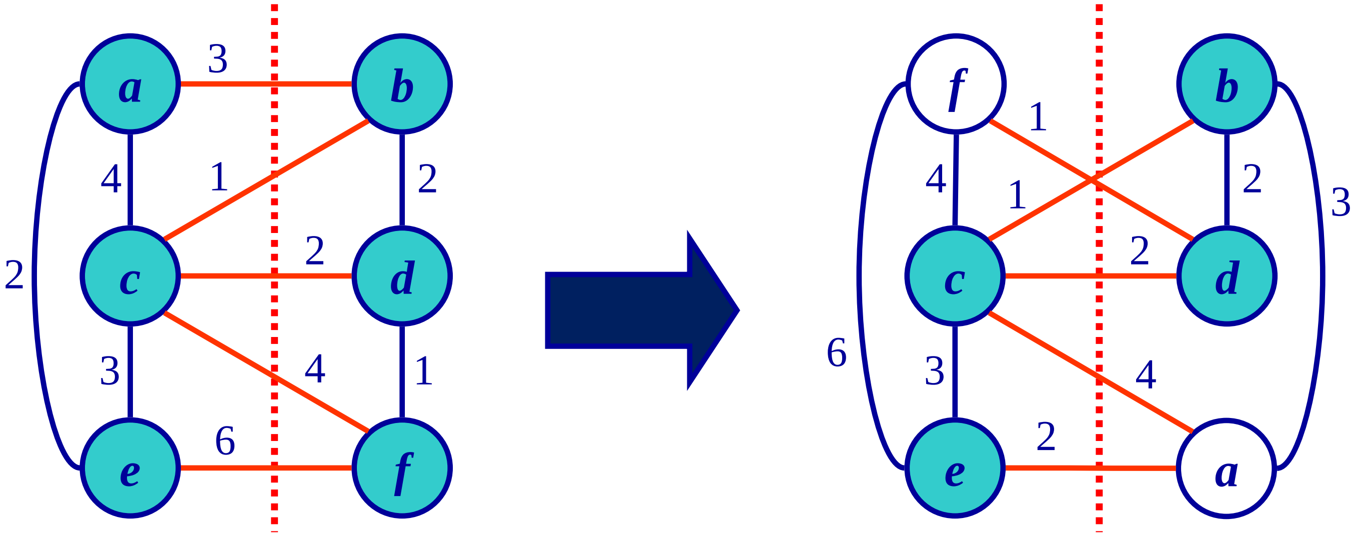
$G_c = \ \ 0$ $\quad G_d = -1$

$G_e = +1$ $\quad G_f = +9$

$$g_{ab} = G_a + G_b - 2c_{ab} = -3 + 2 - 2{\cdot}3 = -7$$
$$g_{ad} = G_a + G_d - 2c_{ad} = -3 - 1 - 2{\cdot}0 = -4$$
$$\boxed{g_{af} = G_a + G_f - 2c_{af} = -3 + 9 - 2{\cdot}0 = +6}$$
$$g_{cb} = G_c + G_b - 2c_{cb} = 0 + 2 - 2{\cdot}1 = 0$$
$$g_{cd} = G_c + G_d - 2c_{cd} = 0 - 1 - 2{\cdot}2 = -5$$
$$g_{cf} = G_c + G_f - 2c_{cf} = 0 + 9 - 2{\cdot}4 = +1$$
$$g_{eb} = G_e + G_b - 2c_{eb} = +1 + 2 - 2{\cdot}0 = +3$$
$$g_{ed} = G_e + G_d - 2c_{ed} = +1 - 1 - 2{\cdot}0 = 0$$
$$g_{ef} = G_e + G_f - 2c_{ef} = +1 + 9 - 2{\cdot}6 = -2$$

Pair with maximum gain

# *Kernighan-Lin Algorithm:*




cut-size = 16

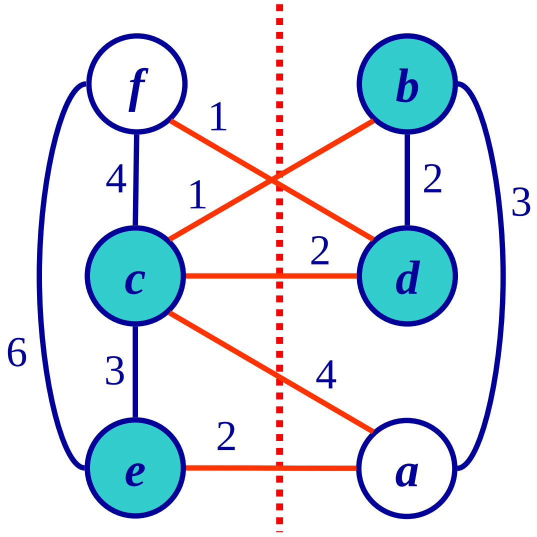
cut-size = 16 – 6 = 10

Exchange nodes *a* and *f*

Then lock up nodes *a* and *f*

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

# *Kernighan-Lin Algorithm:*




cut-size = 10

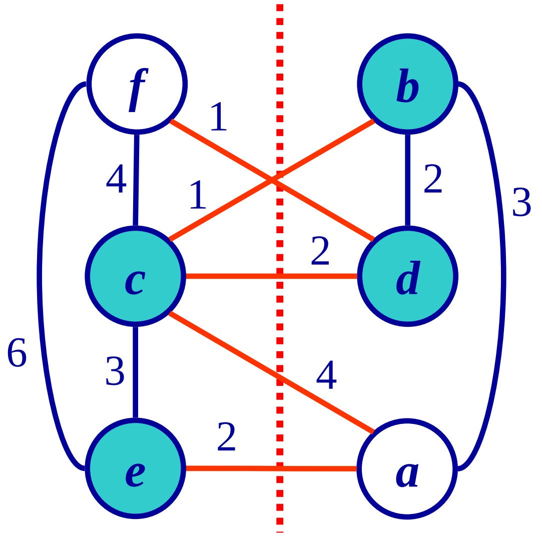
$X' = \{ c, e \}$
$Y' = \{ b, d \}$

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

Update the $G$-values of unlocked nodes

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

# *Kernighan-Lin Algorithm:*




cut-size = 10

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

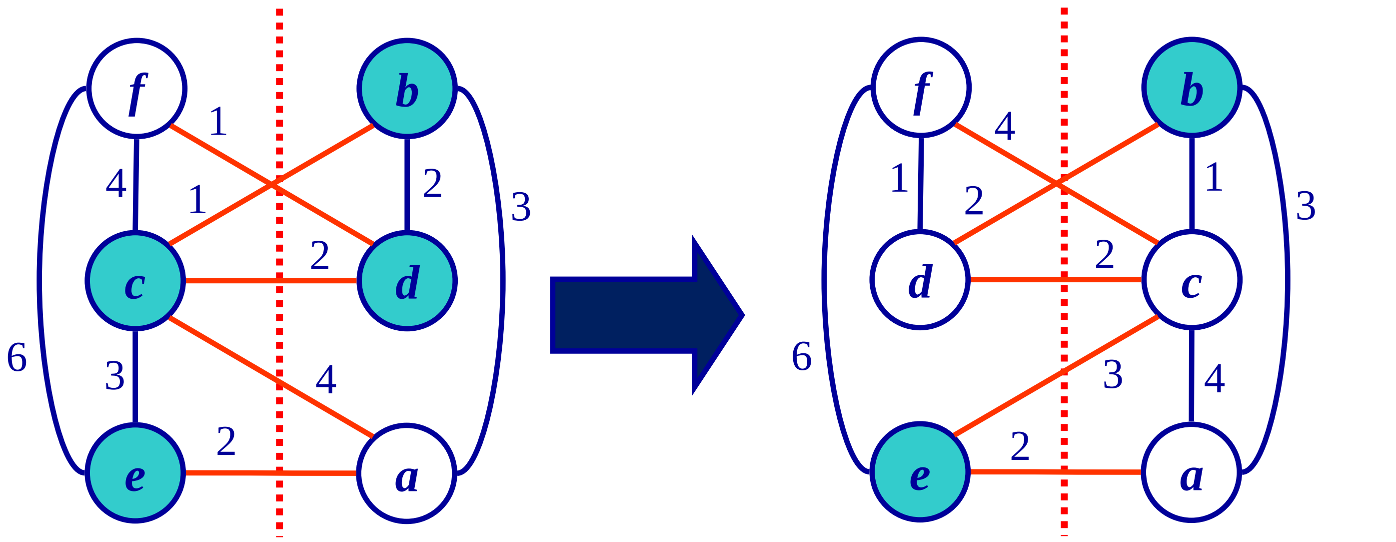
$G'_c = 0 \qquad G'_b = -4$
$G'_e = -7 \qquad G'_d = +1$

Compute the gains

$g'_{cb} = G'_c + G'_b - 2c_{cb} = 0 - 4 - 2 \cdot 1 = -6$
$\boxed{g'_{cd} = G'_c + G'_d - 2c_{cd} = 0 + 1 - 2 \cdot 2 = -3}$
$g'_{eb} = G'_e + G'_b - 2c_{eb} = -7 - 4 - 2 \cdot 0 = -11$
$g'_{ed} = G'_e + G'_d - 2c_{ed} = -7 + 1 - 2 \cdot 0 = -6$

Pair with maximum gain
(can also be negative)

# *Kernighan-Lin Algorithm:*




cut-size = 10

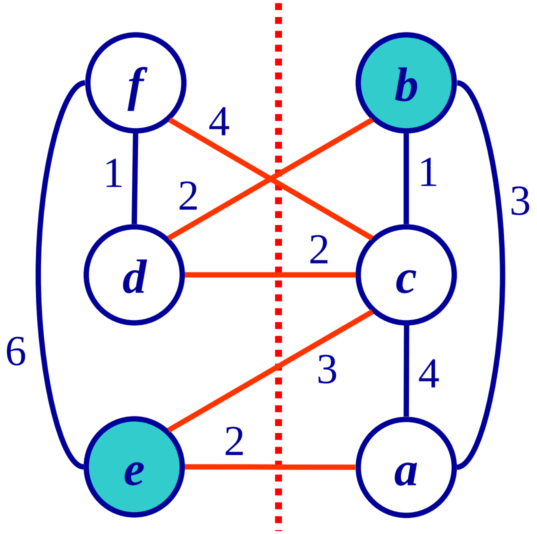
cut-size = 10 – (–3) = 13

Exchange nodes *c* and *d*

Then lock up nodes *c* and *d*

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

# *Kernighan-Lin Algorithm:*




cut-size = 13

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

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

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 ($e$, $b$)

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

# *Kernighan-Lin Algorithm:*

- **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.
- *Repeat the Kernighan-Lin.*

# *Kernighan-Lin Algorithm:*

## *Time Complexity of KL:*

- For each pass
  - $O(n^2)$ time to find the best pair to exchange.
  - $n$ pairs exchanged.
  - Total time is $O(n^3)$ per pass.
- Better implementation can get $O(n^2 \ln n)$ time per pass.
- Number of passes is usually small.

## *Spectral Partitioning Method:*

- How to define a "good" partition of a graph?
  - *Minimize a given graph cut criterion*

- How to efficiently identify such a partition?
  - *Approximate using information provided by the eigenvalues and eigenvectors of a graph*

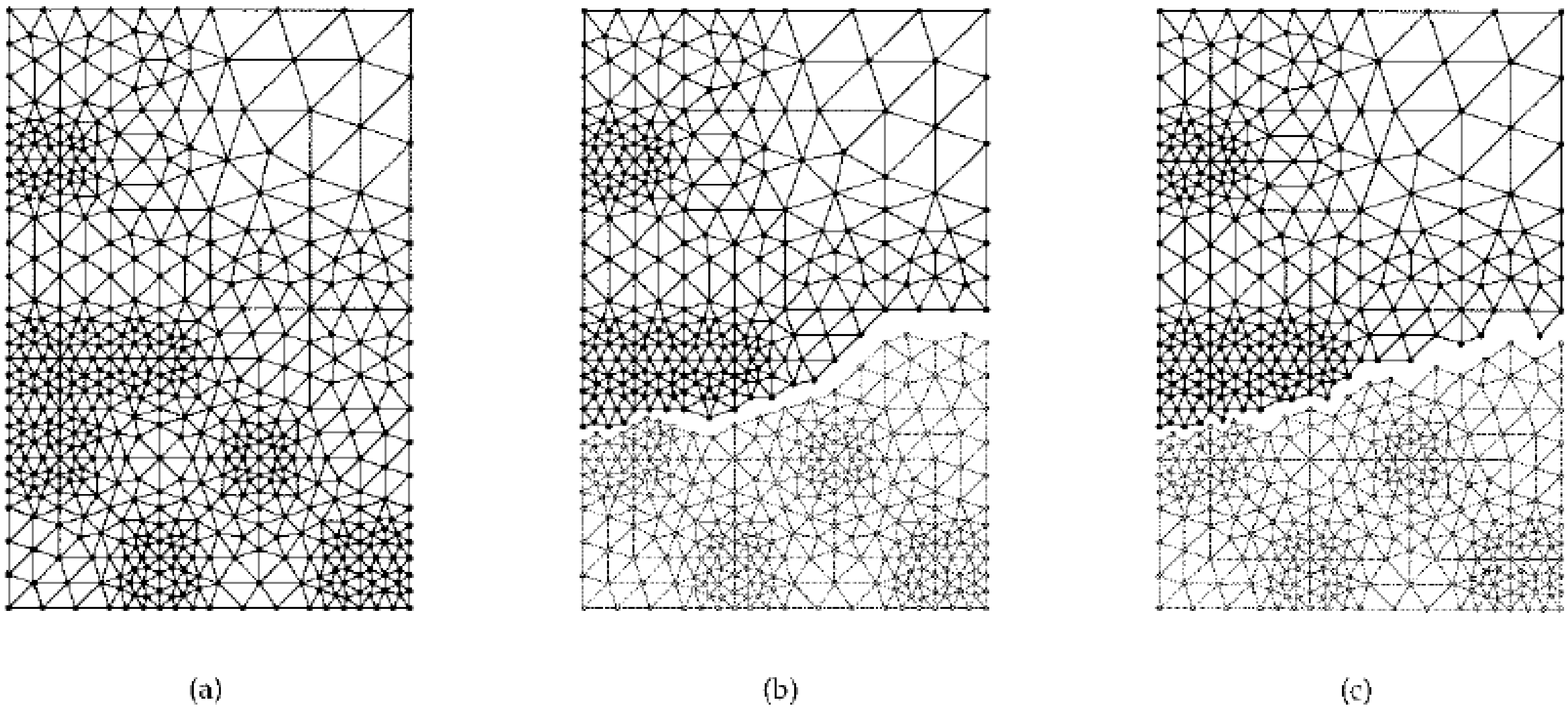
- Spectral Clustering

# *Spectral Partitioning Method:*

- ***Three basic stages:***
  - **1)** *Pre-processing*
    - Construct the Laplacian matrix ***L*** of the graph
  - **2)** *Decomposition*
    - Compute eigenvalues and eigenvectors of the matrix
    - Map each point to a lower-dimensional representation based on one or more eigenvectors
  - **3)** *Grouping*
    - Assign points to two or more clusters, based on the new representation

Fiedler, M., *Algebraic connectivity of graphs*, Czech. Math. 1- 23,298-305 (1973).
Pothen, A., Simon, H., and Liou, K.-P., *Partitioning sparse matrices with eigenvectors of graphs*, SIAM J. Matrix Anal. Appl. 11,430--452 (1990).

# Graph Partitioning




***Graph partitioning applied to a small mesh network.*** (a) A mesh network of 547 vertices of the kind common1y used in finite element analysis. (b) The edges removed indicate the best division of the network into parts of 273 and 274 vertices found by the Kernighan-Lin algorithm. (c) The best division found by spectral partitioning.

# *Graph Partitioning*

## ***To summarize:***

***Graph partitioning*** is not a good way for finding communities, since one has to know in advance how many partitions one has got (in the example above, we just considered $g = 2$). We will see that the approaches in the above are inspiring for the following.

We will discuss the ***Girvan-Newman approach***, that, again to not directly tackle the problem of defining what a community is, but focus on some related properties of the whole structure of the network.

# Community Detection

## *Communities*

- Community: "subsets of actors among whom there are relatively strong, direct, intense, frequent or positive ties."
  - -- Wasserman and Faust, *Social Network Analysis, Methods and Applications*
- Community is a set of actors interacting with each other *frequently*
  - a.k.a. group, subgroup, module, cluster
- A set of people without interaction is NOT a community

# *Community Detection*

## *Communities: Is there a definition?*

The research on community detection is elusive:

*There is not even an agreed definition of communities*

Scientists generally agree that communities must have the following properties

- Communities are connected subgraphs
- Nodes must be more densely connected between a community than across different communities
- Communities reveal the rich interplay between structure and function of a network

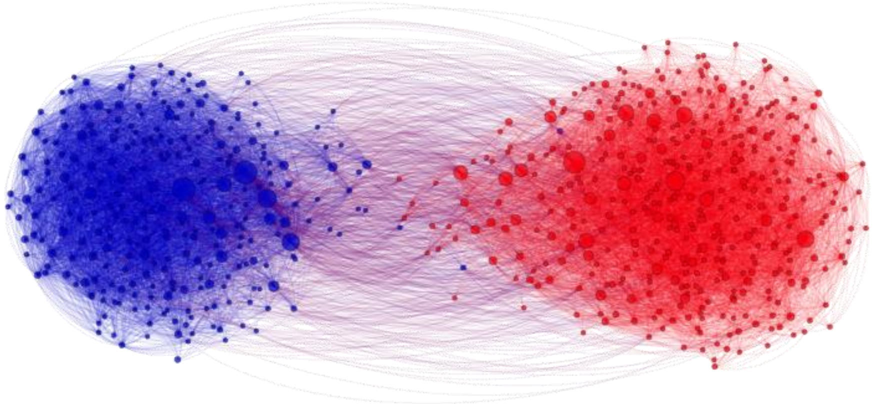
# *Communities: Why analyze?*

**Analyzing communities helps better understand users**

- Users form groups based on their interests

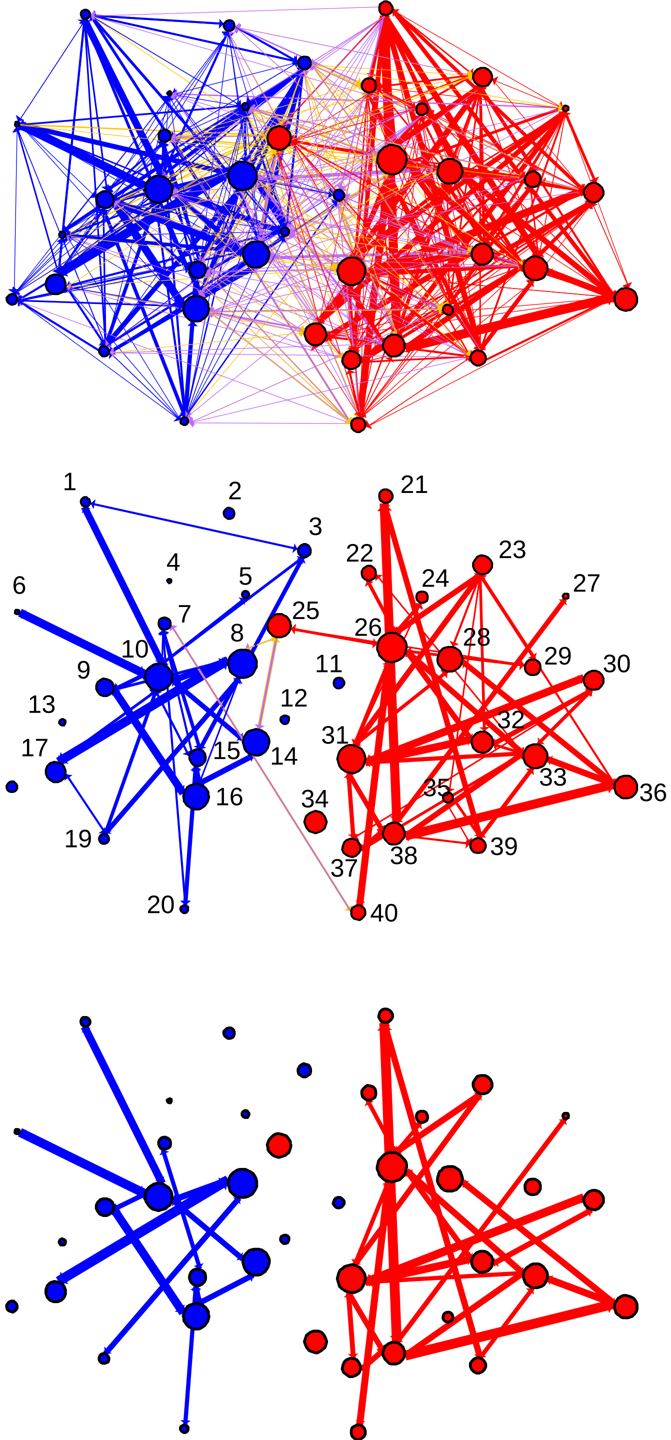
**Groups provide a clear global view of user interactions**

- E.g., find polarization



**Some behaviors are only observable in a group setting and not on an individual level**

– Some republican can **agree** with some democrats, but their parties can **disagree**




(A)

(B)

(C)

1 Digbys Blog
2 James Walcott
3 Pandagon
4 blog.johnkerry.com
5 Oliver Willis
6 America Blog
7 Crooked Timber
8 Daily Kos
9 American Prospect
10 Eschaton
11 Wonkette
12 Talk Left
13 Political Wire
14 Talking Points Memo
15 Matthew Yglesias
16 Washington Monthly
17 MyDD
18 Juan Cole
19 Left Coaster
20 Bradford DeLong

21 Jawa Report
22 Voka Pundit
23 Roger L Simon
24 Tim Blair
25 Andrew Sullivan
26 Instapundit
27 Blogs for Bush
28 Little Green Footballs
29 Belmont Club
30 Captain's Quarters
31 Powerline
32 Hugh Hewitt
33 INDC Journal
34 Real Clear Politics
35 Winds of Change
36 Allahpundit
37 Michelle Malkin
38 WizBang
39 Dean's World
40 Volokh

# Example: political blogs
## (Aug 29$^{th}$ – Nov 15$^{th}$, 2004)

A) all citations between A-list blogs in 2 months preceding the 2004 election

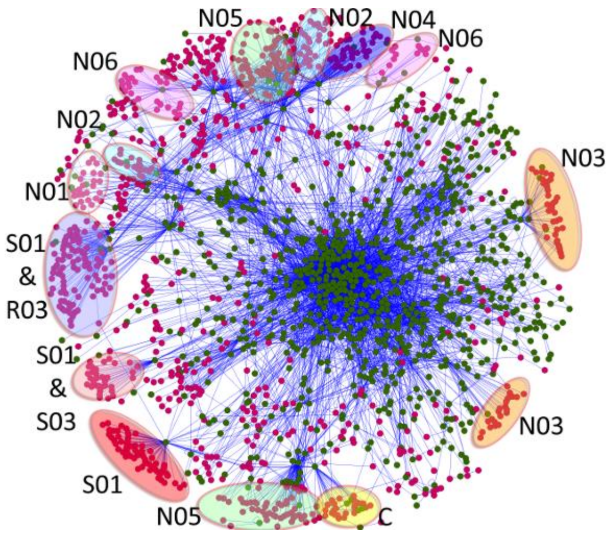
B) citations between A-list blogs with at least 5 citations in both directions

C) edges further limited to those exceeding 25 combined citations

*only 15% of the citations bridge communities*

source: Adamic & Glance, LinkKDD2005

# *Communities: Implicit in other domains*




## Protein-protein interaction networks

- Communities are likely to group proteins having the same specific function within the cell

## World Wide Web

- Communities may correspond to groups of pages dealing with the same or related topics

## Metabolic networks

- Communities may be related to functional modules such as cycles and pathways

## Food webs

- Communities may identify compartments

# *Example: Communities in Metabolic Networks*

***Community structure of the E. coli metabolism***

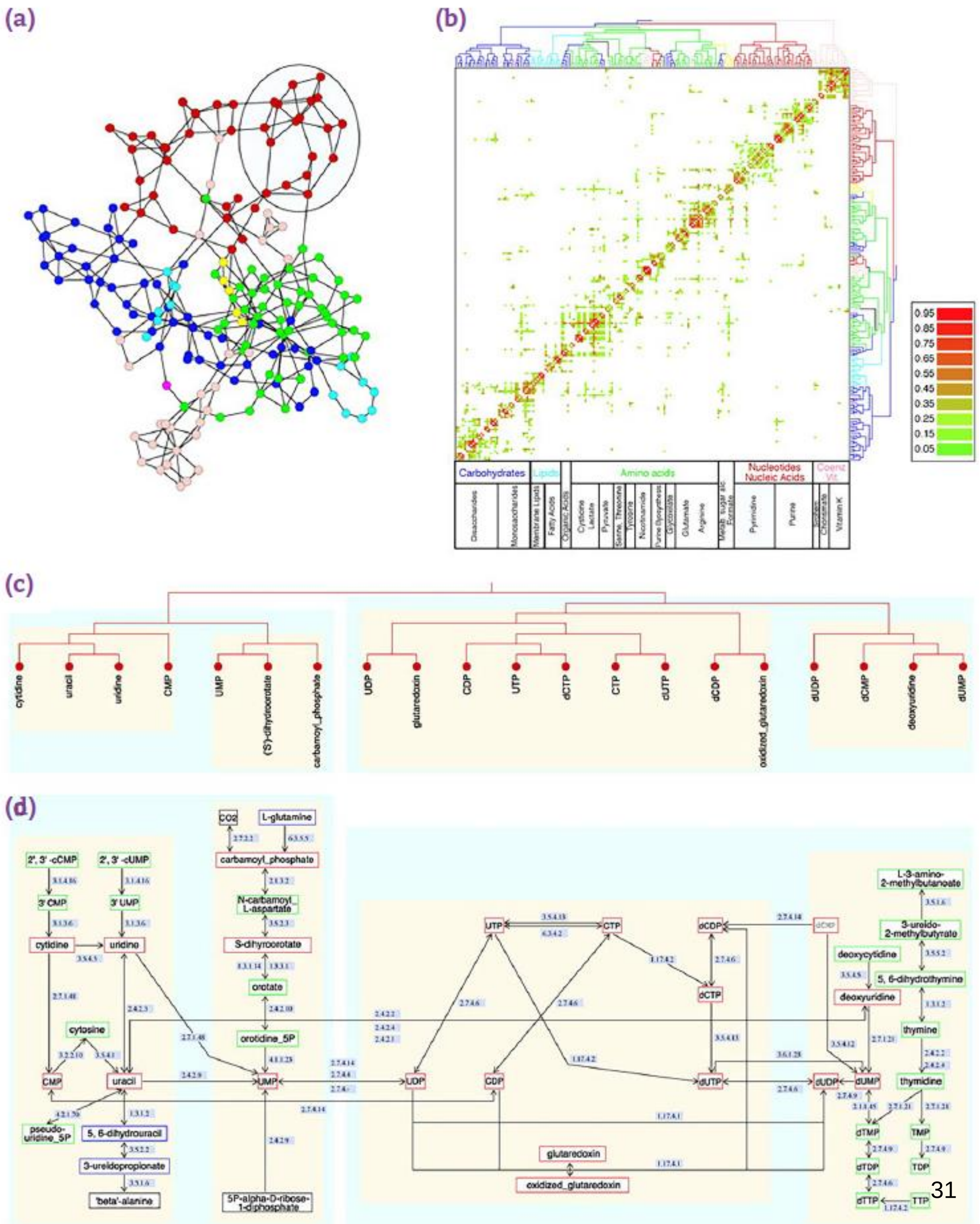
(a) The biological modules (communities) . The color of each node, capturing the predominant biochemical class to which it belongs, indicates that different functional classes are segregated in distinct network neighborhoods. The highlighted region selects the nodes that belong to the pyrimidine metabolism, one of the predicted communities.

(b) The topological overlap matrix of the E. coli metabolism and the corresponding dendrogram that allows one to identify the modules shown in (a). The color of the branches reflect the predominant biochemical role of the participating molecules, like carbohydrates (blue), nucleotide and nucleic acid metabolism (red), and lipid metabolism (cyan).

(c) The red right branch of the dendrogram tree shown in (b), highlighting the region corresponding to the pyridine module.

(d) The detailed metabolic reactions within the pyrimidine module. The boxes around the reactions highlight the communities.

E. Ravasz, A. L. Somera, D. A. Mongru, Z. N. Oltvai, and A.-L. Barabási. *Hierarchical organization of modularity in metabolic networks*. Science, 297:1551-1555, 2002.



# Community Detection

- *Community Detection*: "formalize the strong social groups based on the social network properties"
  - a.k.a. grouping, clustering, finding cohesive subgroups

  - ***Given***: a social network
  - ***Output***: community membership of (some) actors

- Some social media sites allow people to join groups
  - Not all sites provide community platform
  - Not all people join groups

# Community Detection

- Network interaction provides rich information about the relationship between users
  - Is it necessary to extract groups based on network topology?
  - Groups are *implicitly* formed
  - Can complement other kinds of information
  - Provide basic information for other tasks

- Applications
  - Understanding the interactions between people
  - Visualizing and navigating huge networks
  - Forming the basis for other tasks such as data mining

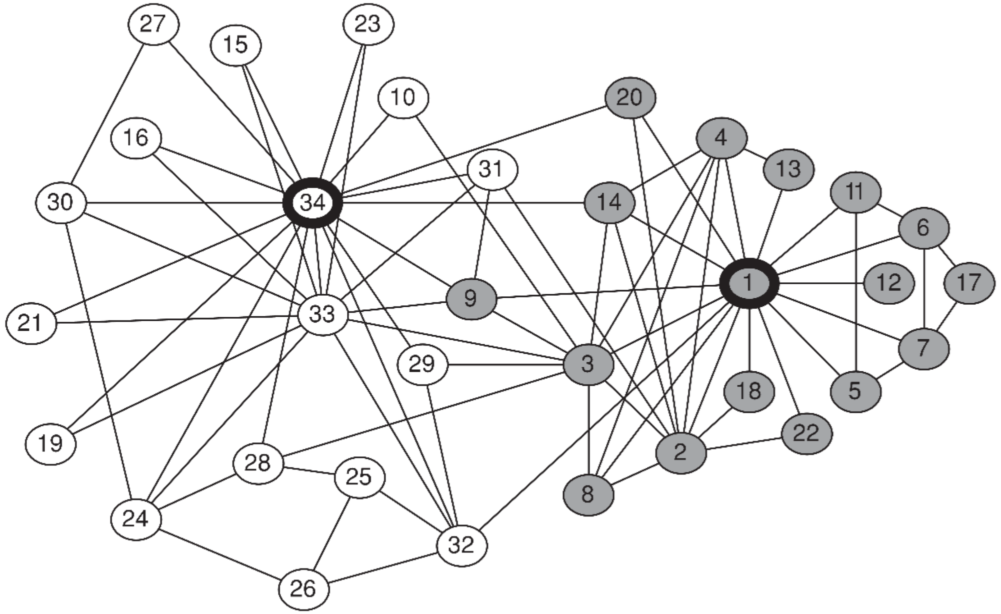
# Community Detection

## *Why is it Important?*

**Zachary's karate club**

*W.W. Zachary, J. Anthropol. Res. 33 (1977) 452.*

Interactions between 34 members of a karate club for over two years

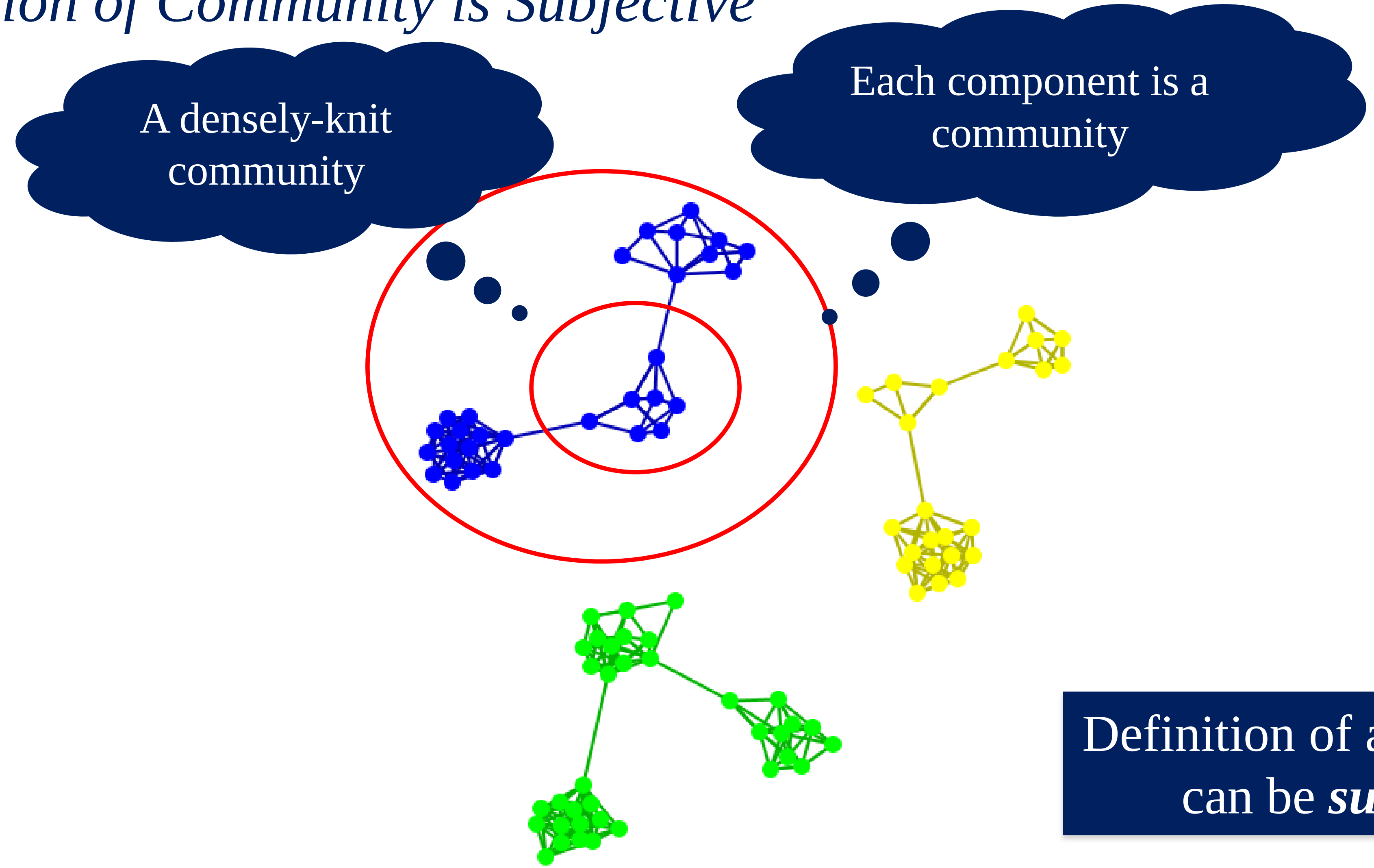



- The club members split into two groups (***gray*** and ***white***)
- Disagreement between the administrator of the club (node **34**) and the club's instructor (node **1**),
- The members of one group left to start their own club

*The same communities can be found by using **community detection***

# Community Detection

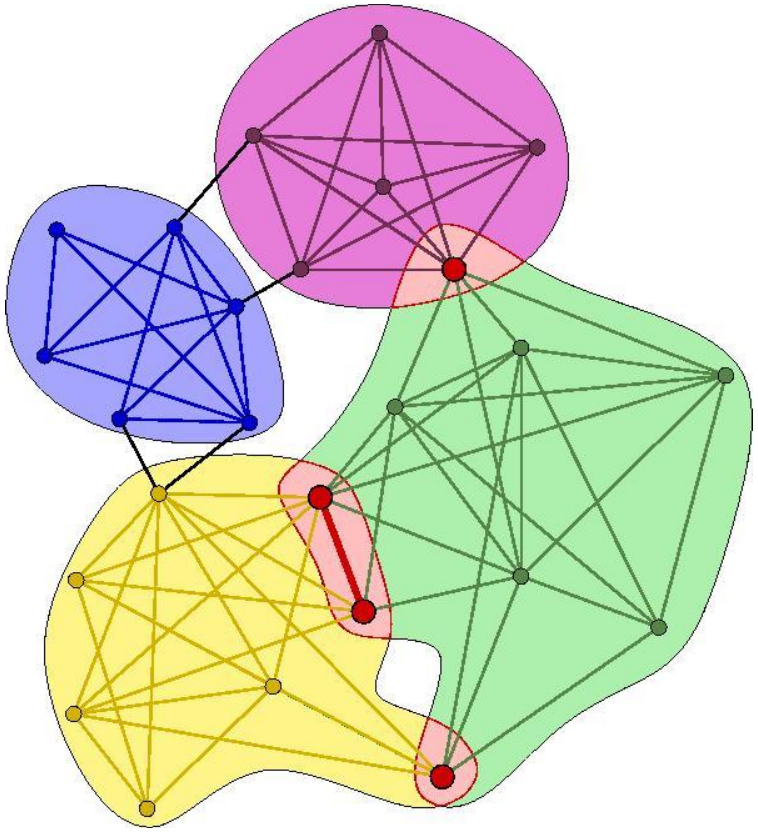
## *Definition of Community is Subjective*



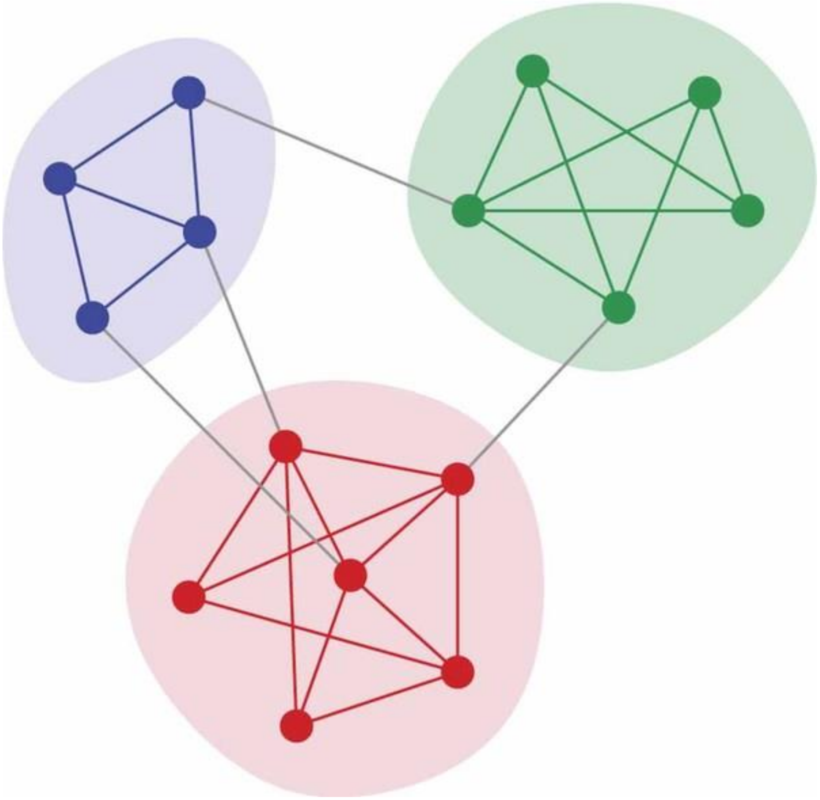

Definition of a community can be ***subjective.***

# Community Detection

## Overlapping vs. Disjoint Communities



Overlapping Communities



Disjoint Communities

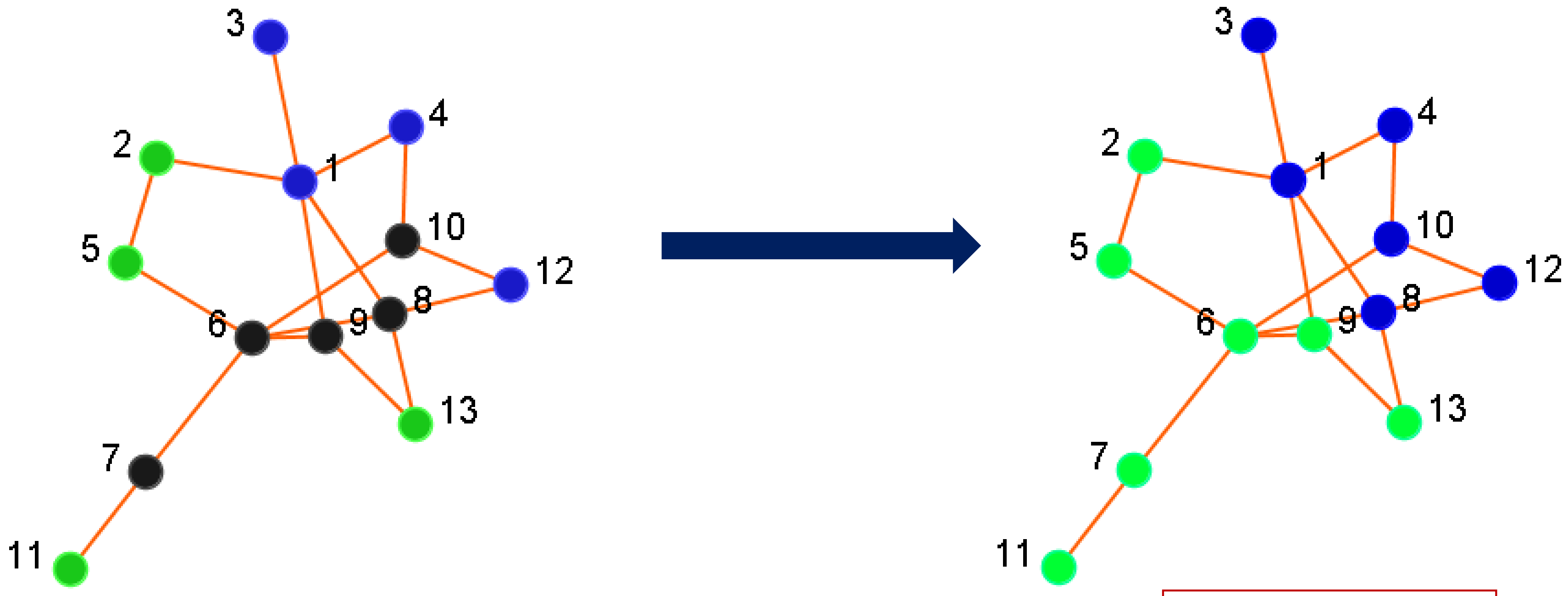
# Community Detection

## *Classification*

- User Preference or Behavior can be represented as class labels
  - Whether or not clicking on an ad
  - Whether or not interested in certain topics
  - Subscribed to certain political views
  - Like/Dislike a product

- Given
  - A social network
  - Labels of some actors in the network
- Output
  - Labels of remaining actors in the network

# Visualization after Prediction

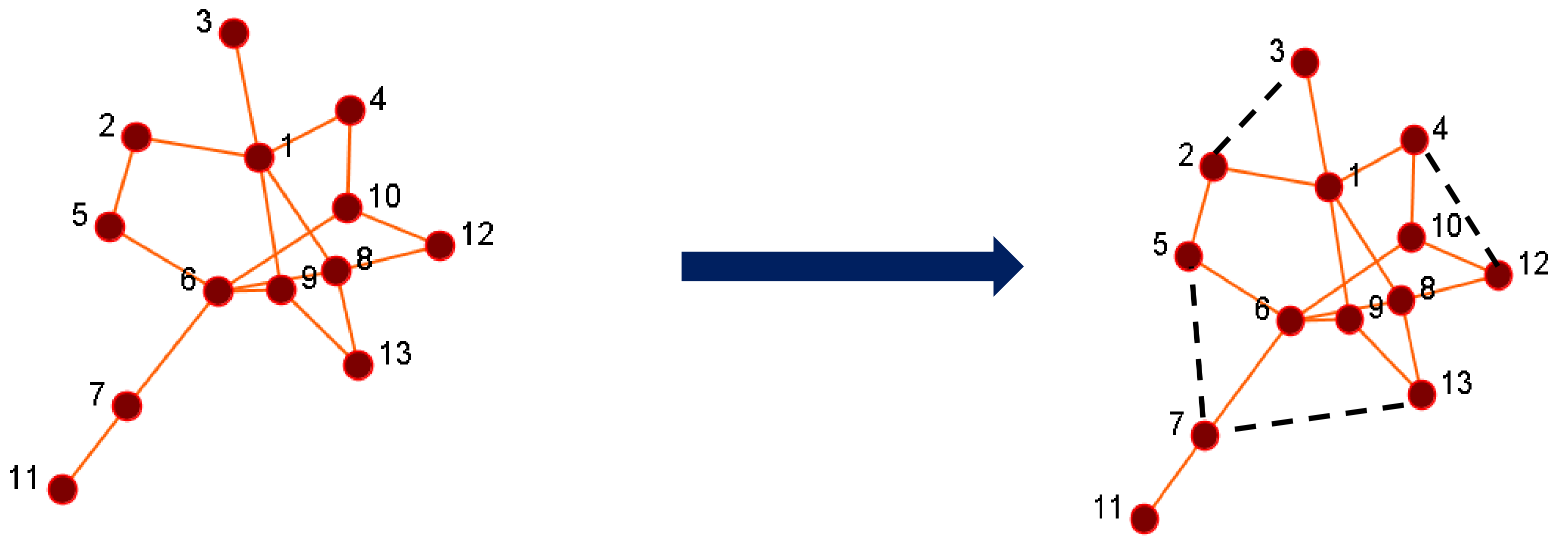



●: Smoking
●: Non-Smoking
●: Unknown?

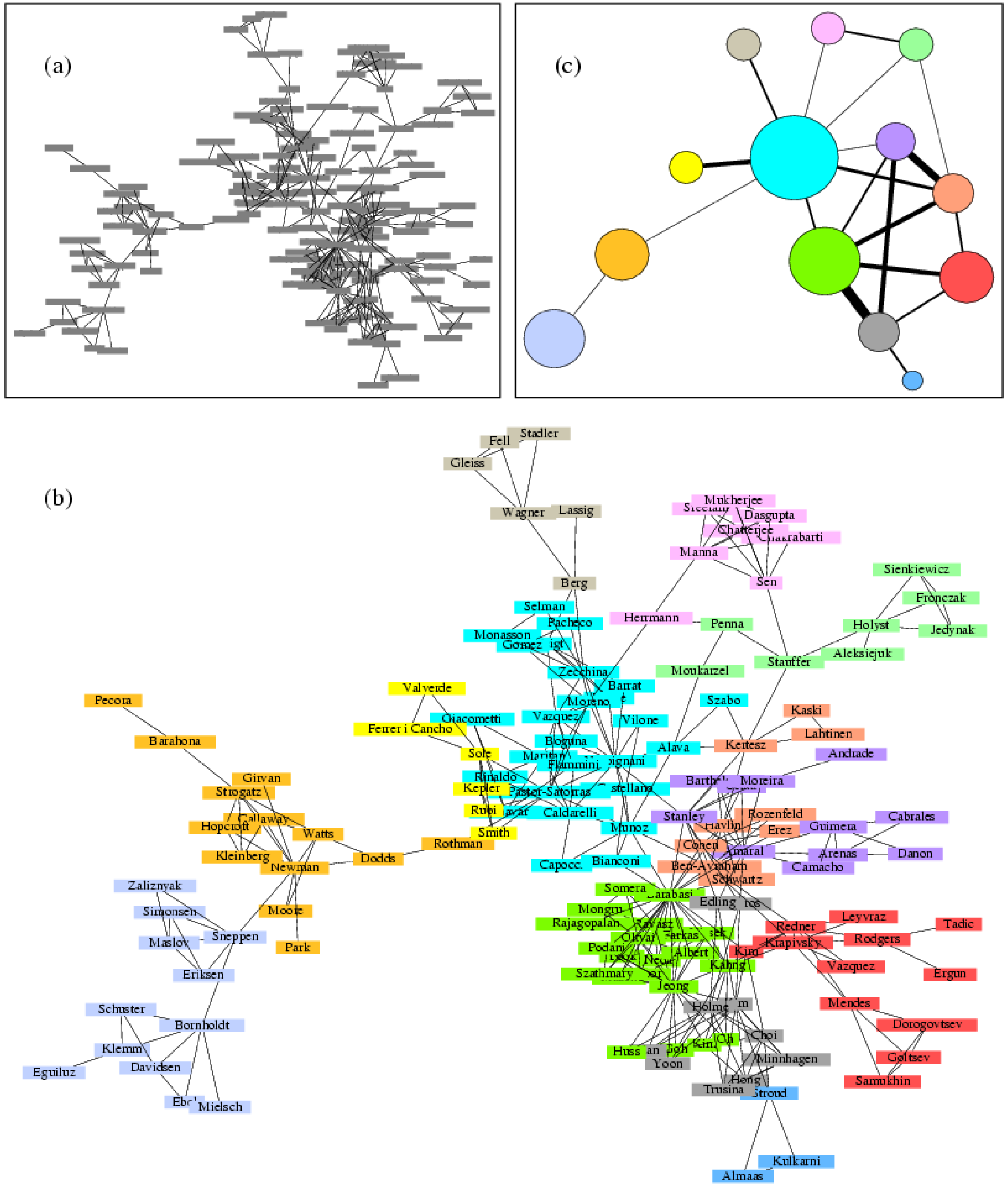
Predictions
6: Non-Smoking
7: Non-Smoking
8: Smoking
9: Non-Smoking
10: Smoking

# Link Prediction

- Given a social network, predict which nodes are likely to get connected
- Output a list of (ranked) pairs of nodes
- Example: Friend recommendation in Facebook




(2, 3)
(4, 12)
(5, 7)
(7, 13)




## *How **Modularity** can help us visualize large networks ?*

# *Modularity Maximization*

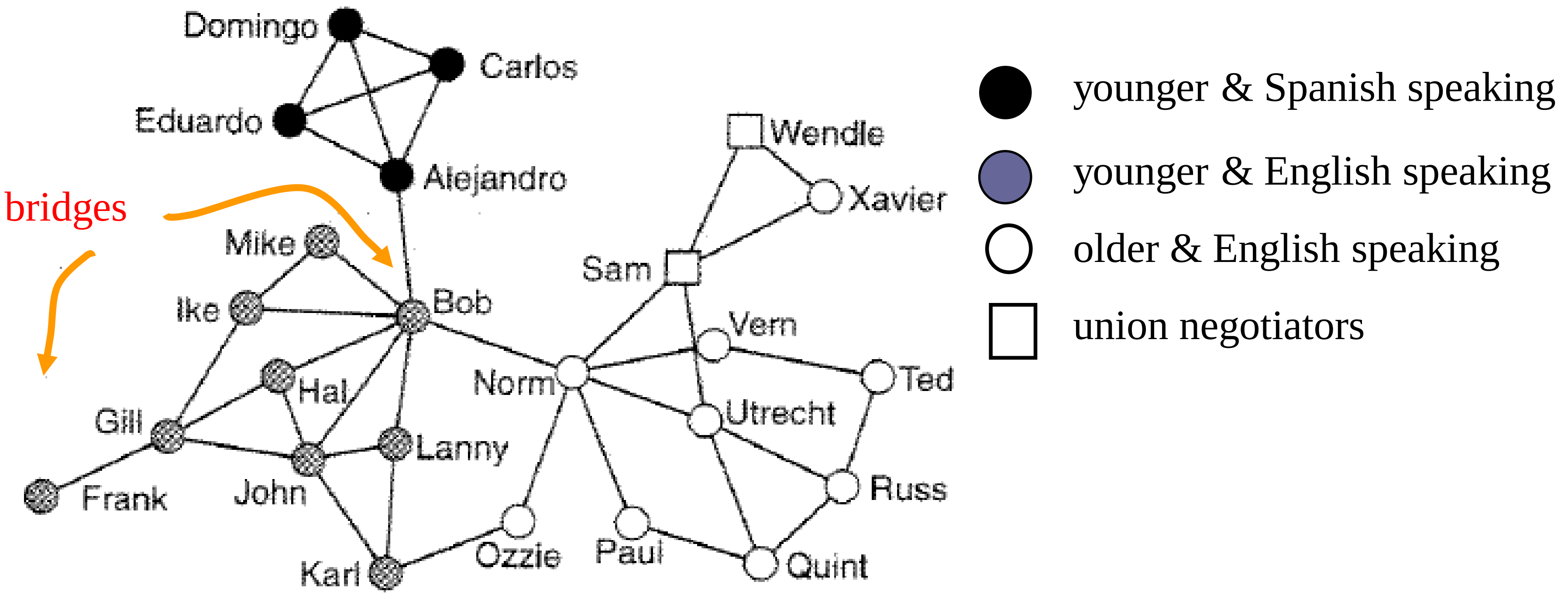
Source: M. E. J. Newman and M. Girvan, Finding and evaluating community structure in networks, Physical Review E 69, 026113 (2004).

# *General properties that indicate Cohesion*

- mutuality of ties
  - everybody in the group knows everybody else
- closeness or reachability of subgroup members
  - individuals are separated by at most n hops
- frequency of ties among members
  - everybody in the group has links to at least k others in the group
- relative frequency of ties among subgroup members compared to nonmembers

# *Social Ties: Bridges*

- Bridge: an edge, that when removed, splits off a community
- Bridges can act as bottlenecks for information flow

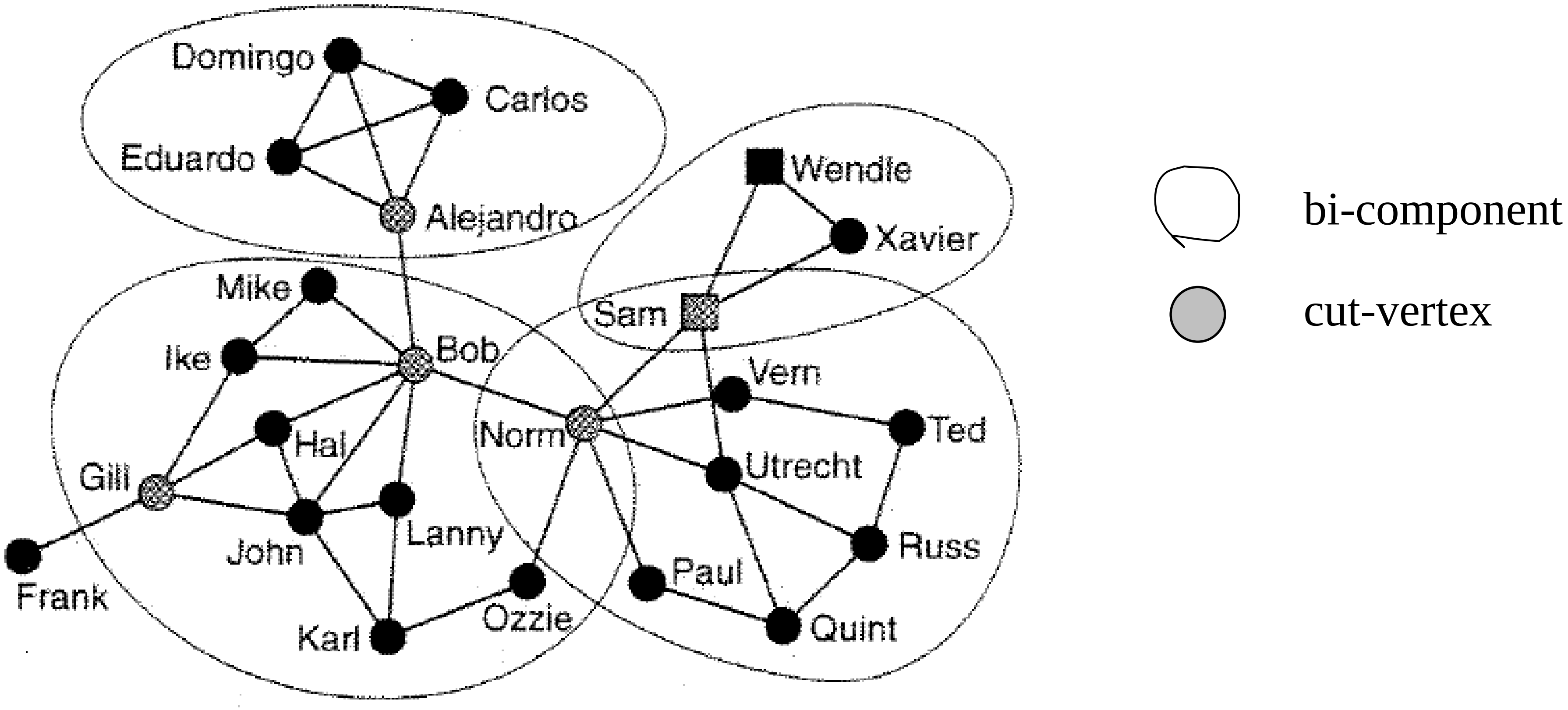



Network of striking employees

de Nooy et al., Exploratory Social Network Analysis with Pajek, Chapter 7, Cambridge U. Press, 2005.

# *Social Ties: Cut-vertices and Bi-components*

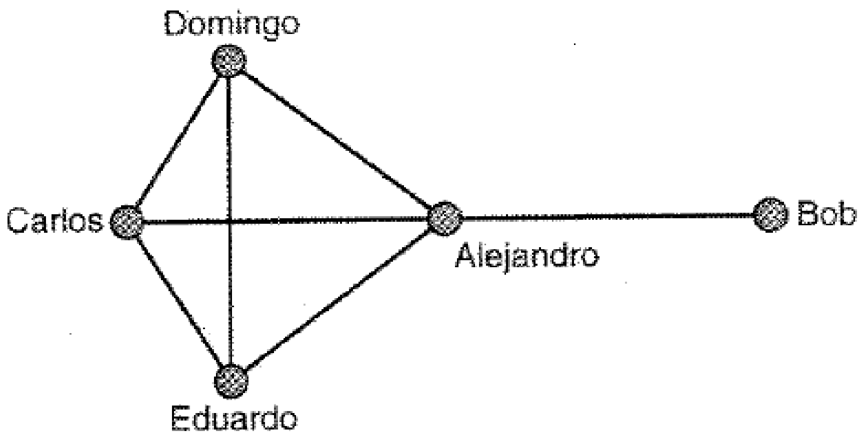
- Removing a cut-vertex creates a separate component
- *bi-component*: component of minimum size 3 that doesn't contain a cut-vertex (vertex that would split the component)




de Nooy et al., Exploratory Social Network Analysis with Pajek, Chapter 7, Cambridge U. Press, 2005.

# *Social Ties: Ego-networks*

***Ego-network***: a *focal node* (“*ego*”), all its neighbors, and connections among the neighbors




Alejandro’s ego-centered network

Alejandro is a ***broker*** between contacts who are not directly connected

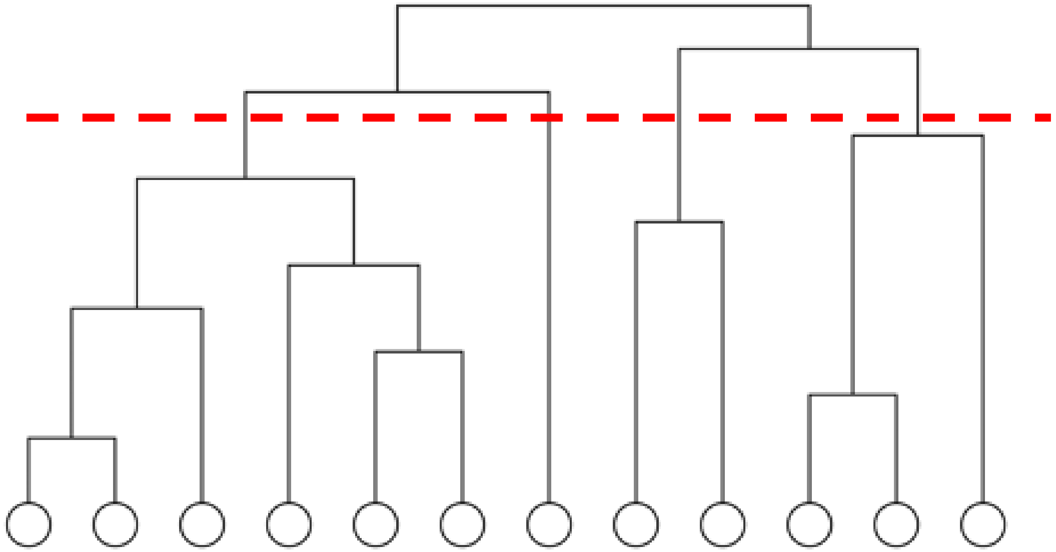
de Nooy et al., Exploratory Social Network Analysis with Pajek, Chapter 7, Cambridge U. Press, 2005.

# *Community Detection: Hierarchical Clustering*

- ***Goal:*** Build a hierarchical structure of communities based on network topology

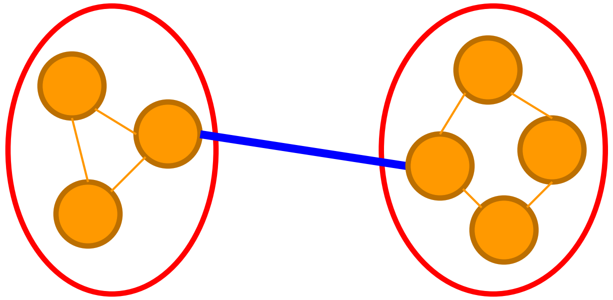
- Facilitate the analysis at different resolutions

- Representative Approaches:
  - Divisive Hierarchical Clustering
  - Agglomerative Hierarchical Clustering



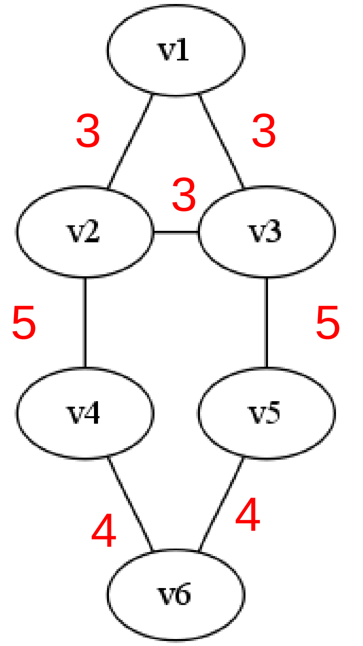
# *Divisive Hierarchical Clustering*

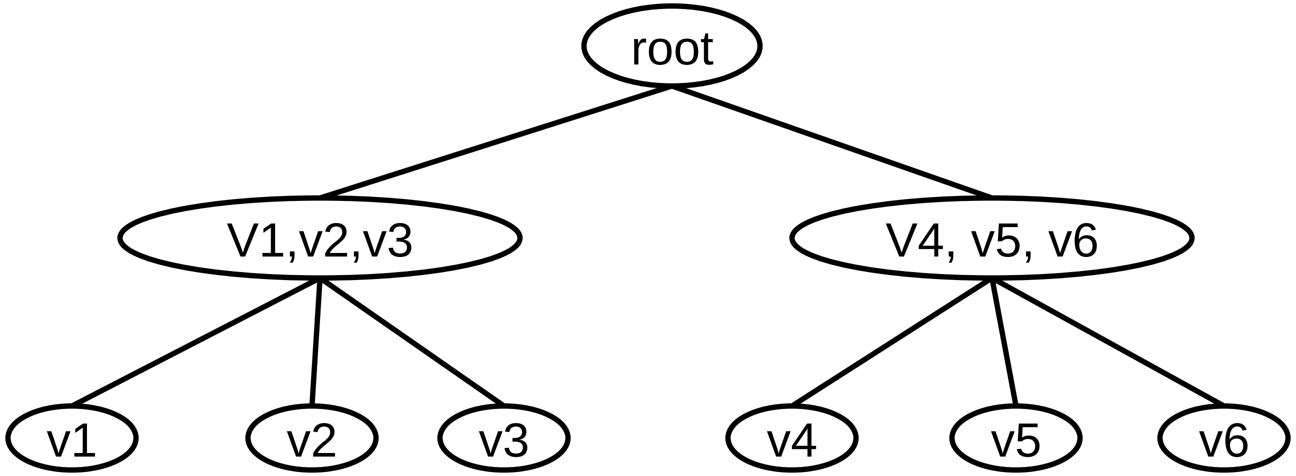
- Divisive Hierarchical Clustering
  - Partition the nodes into several sets
  - Each set is further partitioned into smaller sets
- Network-centric methods can be applied for partition
- One particular example is based on edge-betweenness
  - ***Edge-Betweenness:*** Number of shortest paths between any pair of nodes that pass through the edge
- Between-group edges tend to have larger edge-betweenness



# *Divisive clustering on Edge-Betweenness*

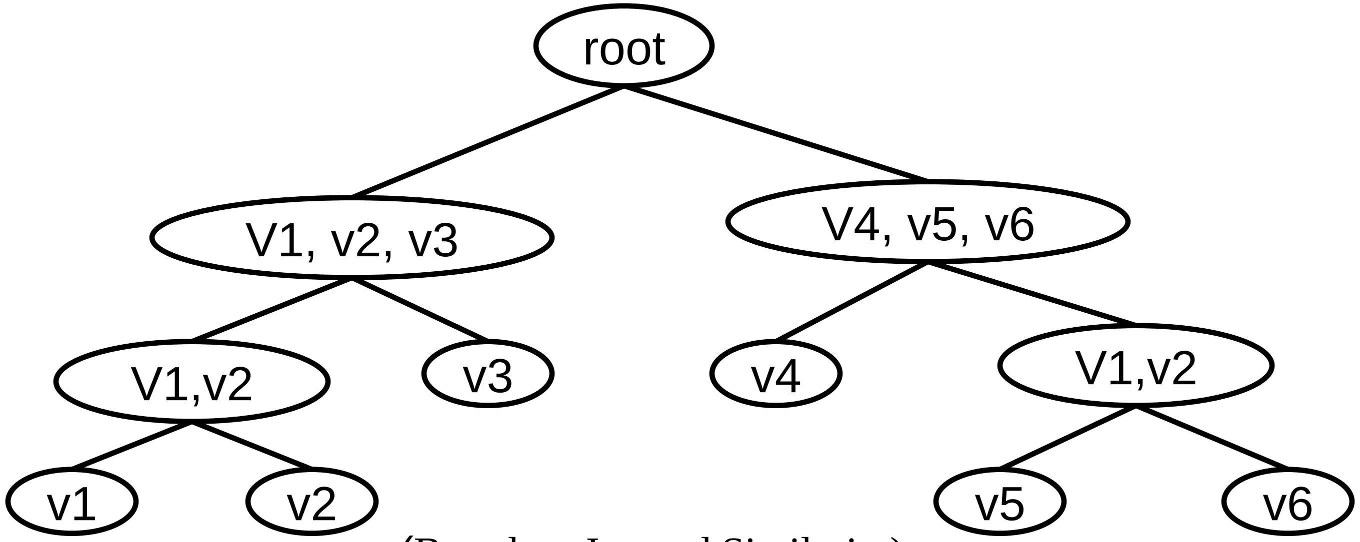
- Progressively remove edges with the highest betweenness
  - Remove e(2,4), e(3, 5)
  - Remove e(4,6), e(5,6)
  - Remove e(1,2), e(2,3), e(3,1)



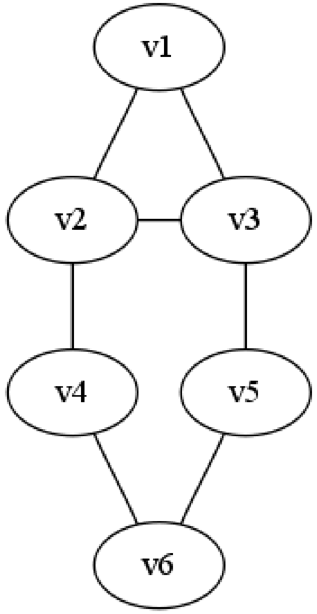



# *Agglomerative Hierarchical Clustering*

- Initialize each node as a community
- Choose two communities satisfying certain *criteria* and merge them into larger ones
  - Maximum Modularity Increase
  - Maximum Node Similarity

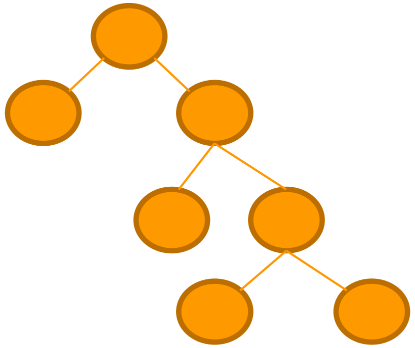



(Based on Jaccard Similarity)



# *Recap of Hierarchical Clustering*

- Most hierarchical clustering algorithm output a binary tree
  - Each node has two children nodes
  - Might be highly imbalanced



- Agglomerative clustering can be very sensitive to the nodes processing order and merging criteria adopted.

- Divisive clustering is more stable, but generally more computationally expensive

# Community Detection: The Girvan-Newman Algorithm

1. Calculate the edge betweenness for all edges in the graph.

2. Remove the edge with the highest betweenness. If there is a tie, choose a random edge.

3. Recalculate betweenness for all edges affected by removing the edge.

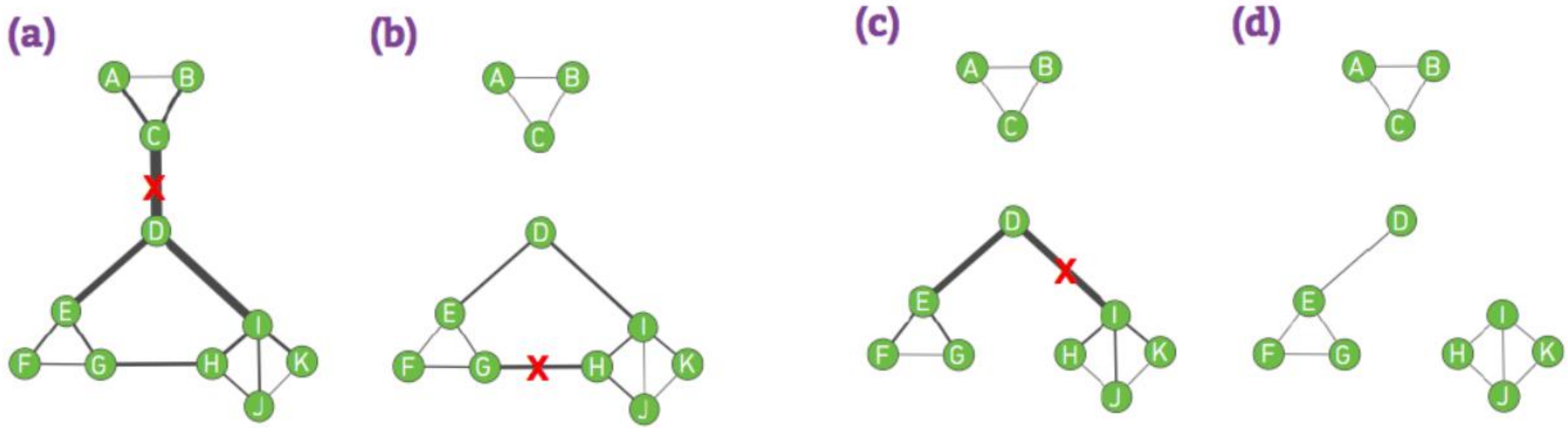
4. Repeat until all edges are removed.

==> It is a *divisive* method

==> What you obtain is a *dendrogram*

*** Takes time of order ~O($n^3$)

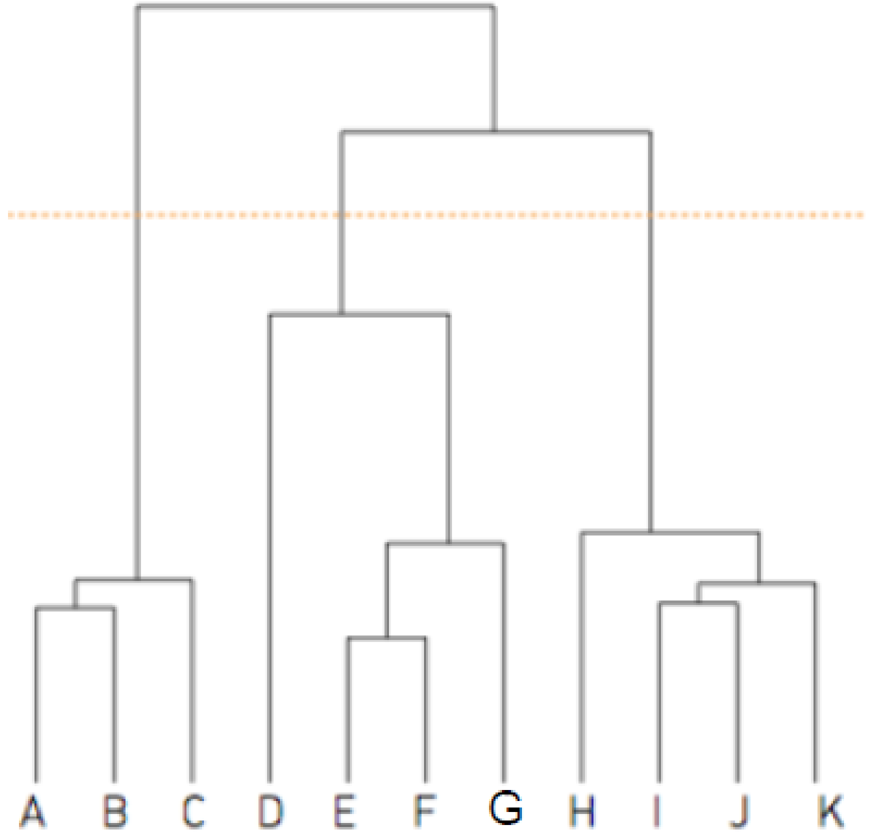
# Community Detection: The Girvan-Newman Algorithm

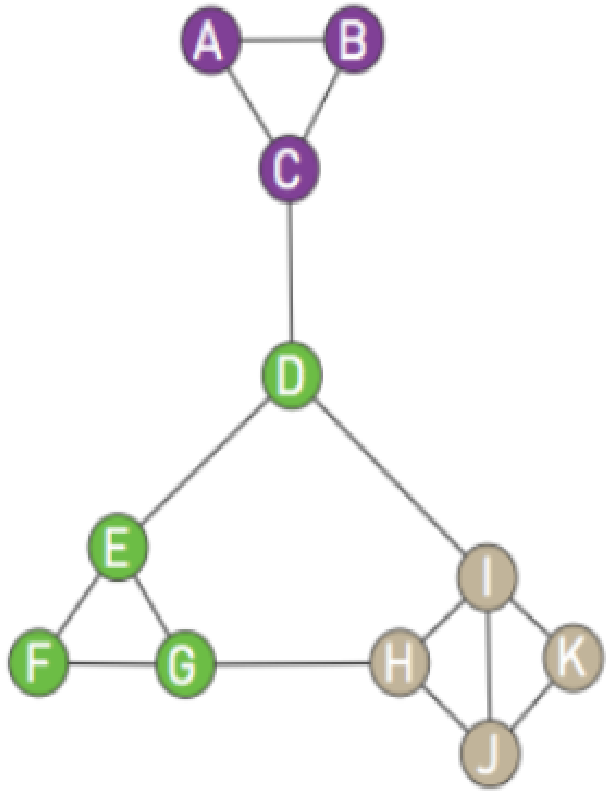


# Community Detection: The Girvan-Newman Algorithm

## *The Dendrogram: Hierarchical Clustering*

[(C,D);(H,G);(D,I);(D,E);(H,I),(H,J),(G,E);(G,F);(K,I);(K,J);(C,A);(C,B);(A,B);(I,J);(E,F)]

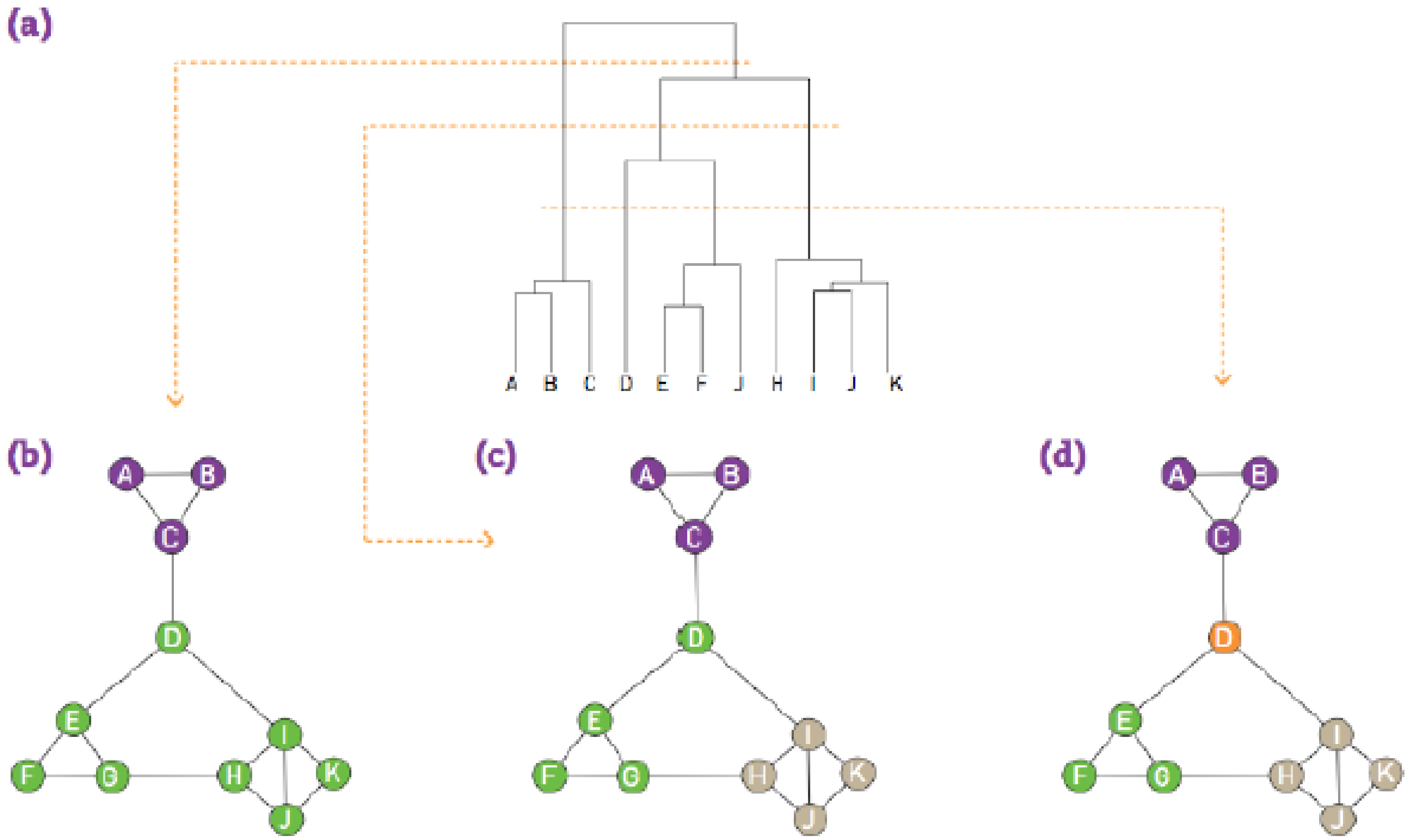


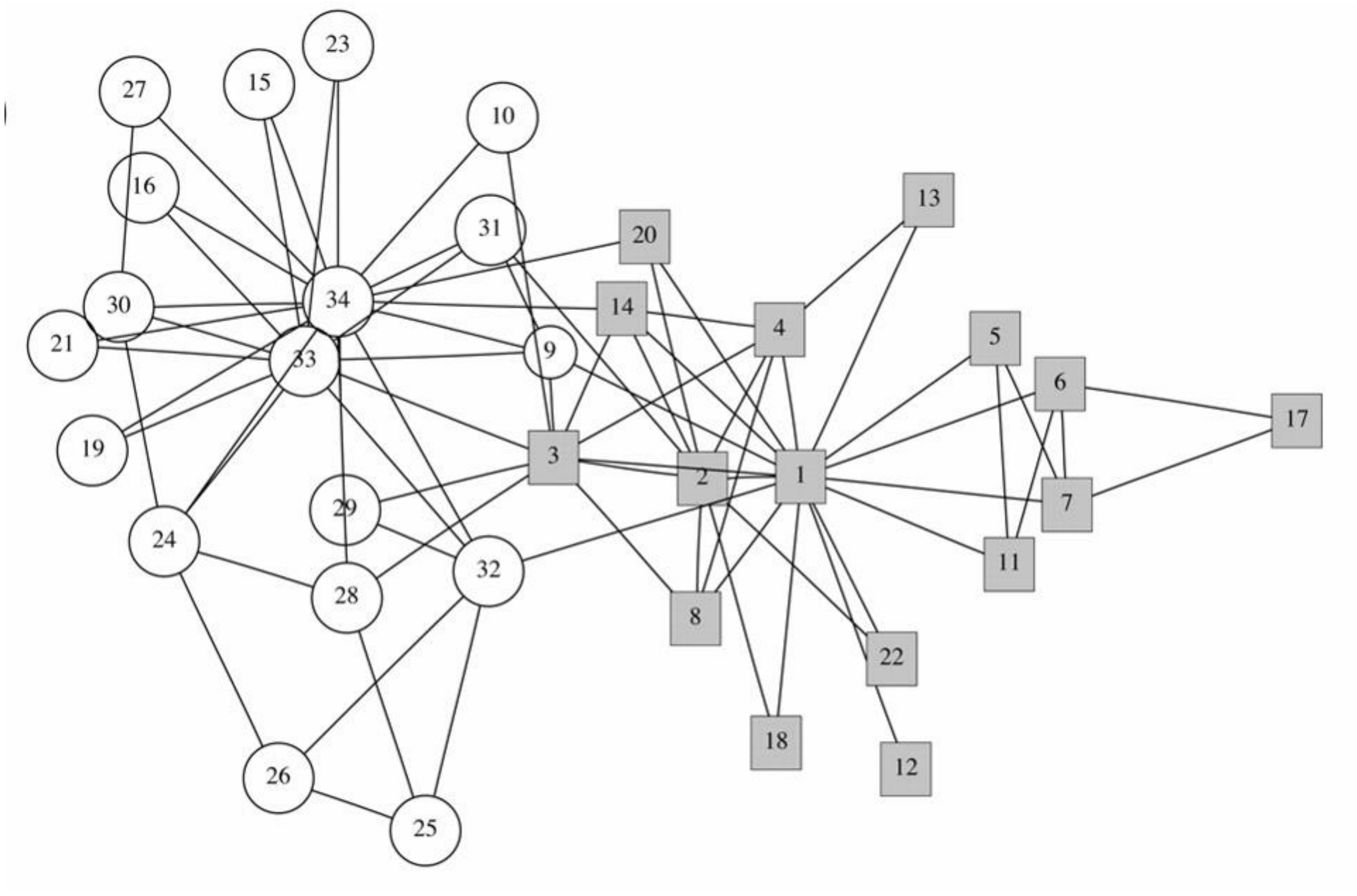

*The dendrogram generated by the Girvan-Newman algorithm. The cut (orange dotted line) reproduces the three communities present in the network.*

# Community Detection: The Girvan-Newman Algorithm

*Where to cut?* → *Modularity Maximization*

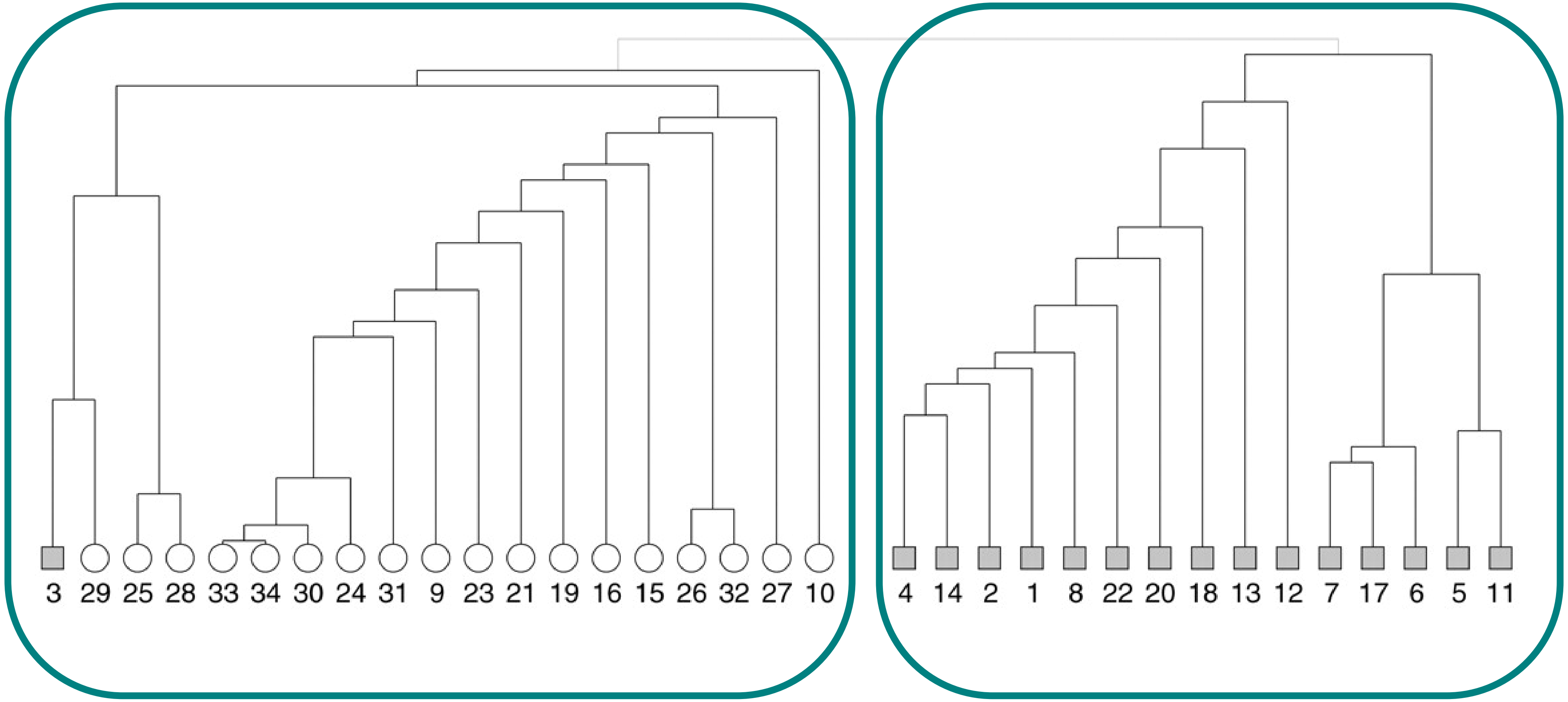


# *Hierarchical clustering: Zachary Karate Club*

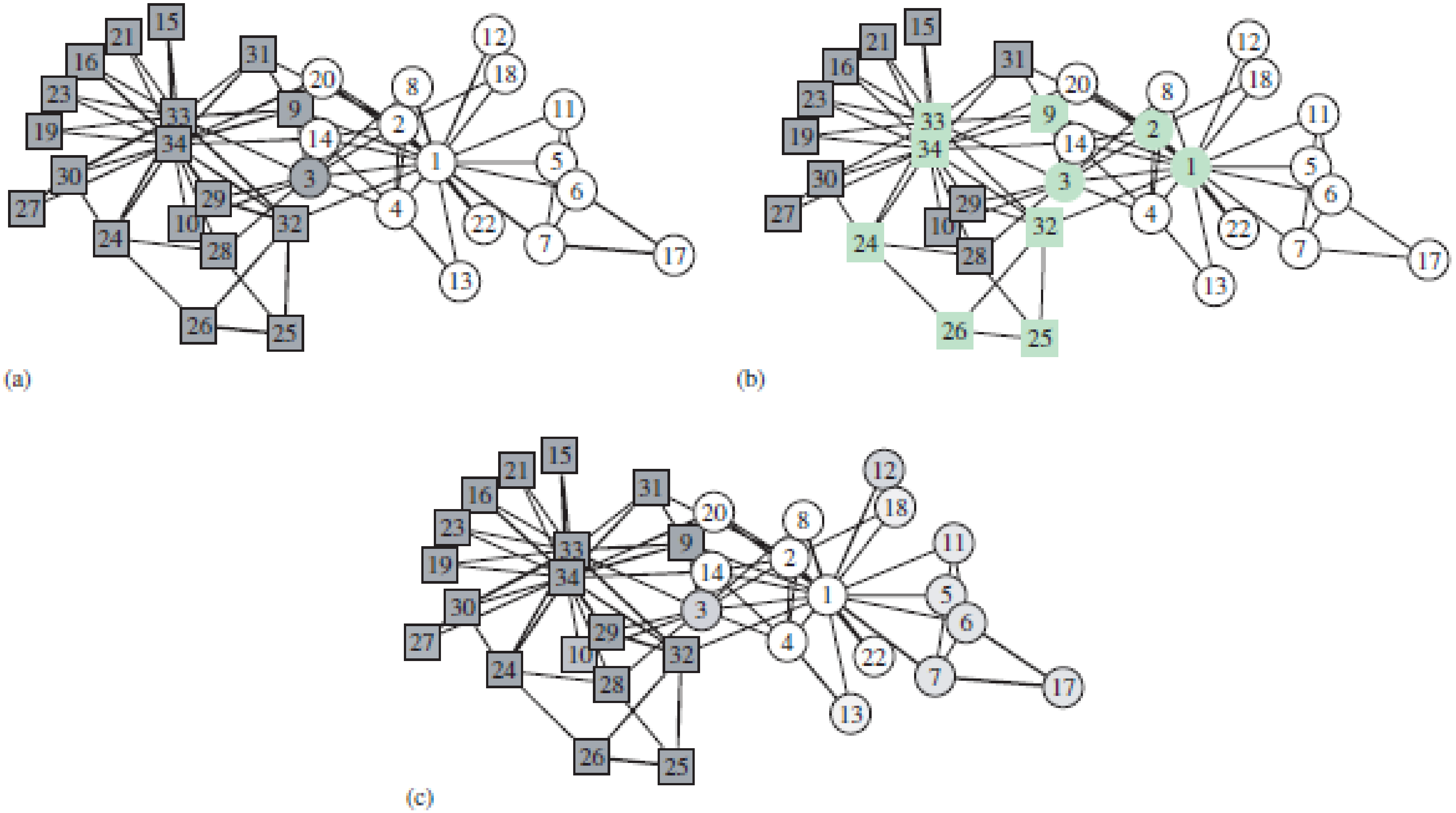



source: Girvan and Newman, PNAS June 11, 2002 99(12):7821-7826

# *Betweenness clustering algorithm & the karate club data set*



# Community Detection




***Finding community structures in the karate club network of Zachary.*** The numbered vertices of the network represent the members of the club, while the edges represent friendships, as determined by the observation of the interactions. The two groups into which the club split during the course of the study are indicated by the squares and circles, while the dark grey and white show the divisions of the network found by (a) the spectral bisection algorithm, (b) the hierarchical clustering method and (c) the Monte Carlo sampled version of the algorithm of Girvan and Newman. In (b) the lightly shaded vertices are those not assigned by the algorithm to either of the two principal communities.  In (c) shades intermediate between the dark grey and white indicate ambiguously assigned vertices that fall in one community or the other, or neither, on different runs of the algorithm.

# *Modularity and Modularity Maximization*

- Given a degree distribution, we know the expected number of edges between any pairs of vertices

- We assume that real-world networks should be far from random.
  - Therefore, the more distant they are from this randomly generated network, the more structural they are.

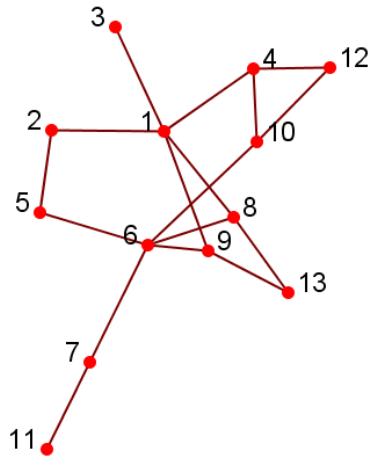
- Modularity defines this distance and modularity maximization tries to maximize this distance

- Example: Louvain Algorithm (optimize modularity directly)

# *Modularity Maximization*

- ***Modularity*** measures the group interactions compared with the *expected random connections* in the group
- In a network with $m$ edges, for two nodes with degree $k_i$ and $k_j$ , expected random connections between them are

$$k_i k_j / 2m$$




- The interaction utility in a group:

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

- To partition the group into multiple groups, we maximize

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

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

# *Modularity Matrix*

- The modularity maximization can be formulated in matrix form

$$Q = \frac{1}{2m}\sum_{ij}\left(A_{ij} - \frac{k_i k_j}{2m}\right)\delta\left(c_i, c_j\right) = \frac{1}{2m}Tr(B\Delta\Delta^T) = \frac{1}{2m}Tr(\Delta^T B\Delta)$$

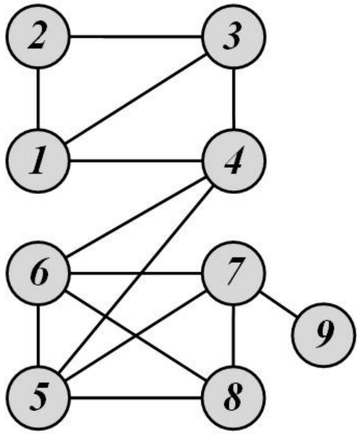
$B_{ij}$ $\quad\quad\quad\quad$ $(\Delta\Delta^T)_{i,j}$

- $B$ is the modularity matrix

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

- ***Solution***: Top eigenvectors of the modularity matrix

# *Modularity Maximization: Example*




$$B = A - kk^T/2m$$

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

$$B = \begin{bmatrix} -0.32 & 0.79 & 0.68 & 0.57 & -0.43 & -0.43 & -0.43 & -0.32 & -0.11 \\ 0.79 & -0.14 & 0.79 & -0.29 & -0.29 & -0.29 & -0.29 & -0.21 & -0.07 \\ 0.68 & 0.79 & -0.32 & 0.57 & -0.43 & -0.43 & -0.43 & -0.32 & -0.11 \\ 0.57 & -0.29 & 0.57 & -0.57 & 0.43 & 0.43 & -0.57 & -0.43 & -0.14 \\ -0.43 & -0.29 & -0.43 & 0.43 & -0.57 & 0.43 & 0.43 & 0.57 & -0.14 \\ -0.43 & -0.29 & -0.43 & 0.43 & 0.43 & -0.57 & 0.43 & 0.57 & -0.14 \\ -0.43 & -0.29 & -0.43 & -0.57 & 0.43 & 0.43 & -0.57 & 0.57 & 0.86 \\ -0.32 & -0.21 & -0.32 & -0.43 & 0.57 & 0.57 & 0.57 & -0.32 & -0.11 \\ -0.11 & -0.07 & -0.11 & -0.14 & -0.14 & -0.14 & 0.86 & -0.11 & -0.04 \end{bmatrix}$$

Modularity Matrix

2 eigenvectors

$$\begin{bmatrix} 0.44 & -0.00 \\ 0.38 & 0.23 \\ 0.44 & -0.00 \\ 0.17 & -0.48 \\ -0.29 & -0.32 \\ -0.29 & -0.32 \\ -0.38 & 0.34 \\ -0.34 & -0.08 \\ -0.14 & 0.63 \end{bmatrix}$$

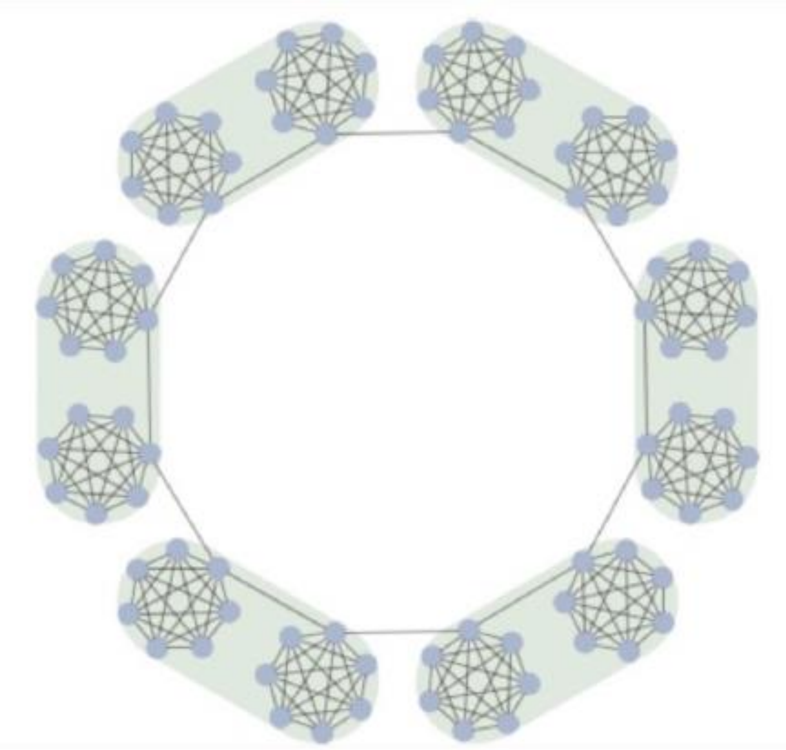
*Two Communities:*
{1, 2, 3, 4}
and
{5, 6, 7, 8, 9}

# *Properties of Modularity*

- ***Properties of modularity:***
  - Between (-1, 1)
  - Modularity = 0, if all nodes are clustered into one group
  - Can automatically determine optimal number of clusters

- ***Resolution limit of modularity***
  - Modularity maximization might return a community consisting multiple small modules
  - Modularity has a "***favorite scale***". For a graph of given *density* and *size*:
    - Communities cannot be smaller than a fraction of nodes
    - Communities cannot be larger than a fraction of nodes
  - Modularity optimization will ***never*** discover
    - Small communities in large networks
    - Large communities in small networks

# *Properties of Modularity*



## ***Resolution limit of modularity (Cont'd)***

Consider a ring of cliques
Cliques are as dense as possible
Single edge between them ==>As separated as possible
Any acceptable algorithm ==>Each clique is a community

But with modularity:
Small graphs ==> OK
Large graphs ==> Maximizing modularity obtained by merging cliques

# Community Detection: Summary

- The Optimal Method?
  - It varies depending on applications, networks, computational resources etc.

- Other lines of research
  - Communities in directed networks
  - Overlapping communities
  - Community evolution
  - Group profiling and interpretation

# *Network and Community Evolution*

- How does a **network** change over time?
- How does a **community** change over time?
- What properties do you expect to remain roughly constant?
- What properties do you expect to change?

- For example,
  - Where do you expect new edges to form?
  - Which edges do you expect to be dropped?