Graph and Network Analysis

Clique

Dr. Derek Greene

Clique Research Cluster, University College Dublin

Web Science Doctoral Summer School 2011

UCD DUBLIN

sfi Science Foundation Ireland

DERI

NUI Galway OÉ Gaillimh

Tutorial Overview

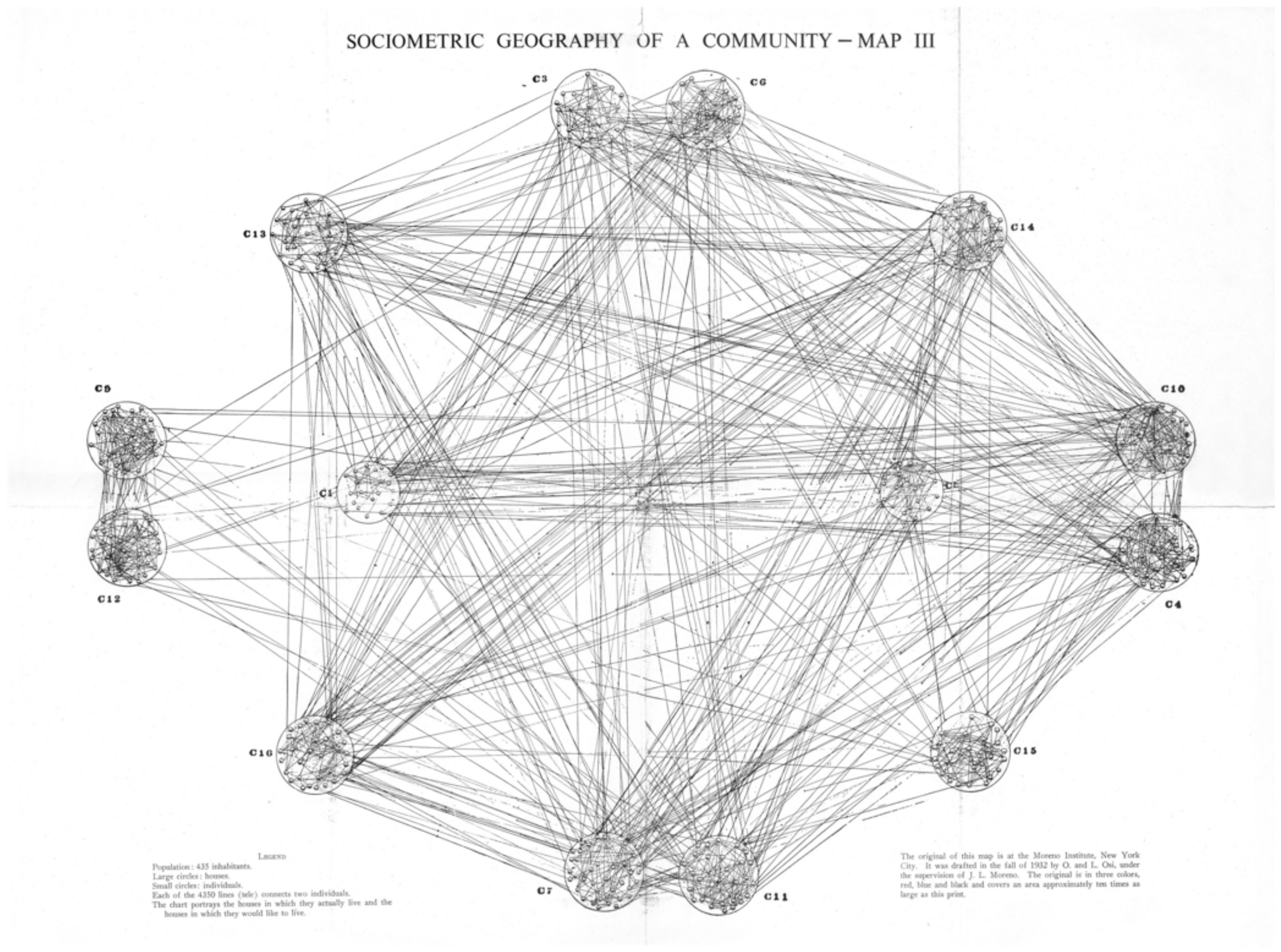
- Practical Network Analysis
  - Basic concepts
  - Network types and structural properties
  - Identifying central nodes in a network
- Communities in Networks
  - Clustering and graph partitioning
  - Finding communities in static networks
  - Finding communities in dynamic networks
- Applications of Network Analysis

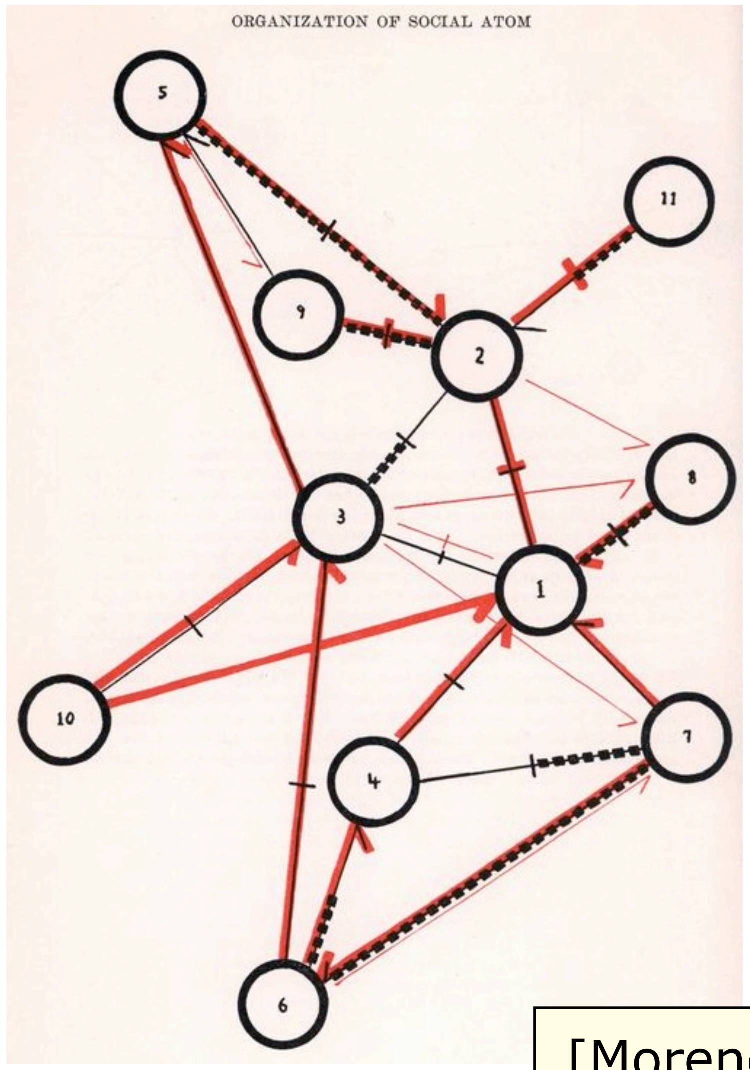
Tutorial Resources

- **NetworkX**: Python software for network analysis (v1.5)
  http://networkx.lanl.gov
- Python 2.6.x / 2.7.x
  http://www.python.org
- **Gephi**: Java interactive visualisation platform and toolkit.
  http://gephi.org
- Slides, full resource list, sample networks, sample code snippets online here:
  http://mlg.ucd.ie/summer

Introduction

- Social network analysis - an old field, rediscovered...





[Moreno,1934]

Introduction

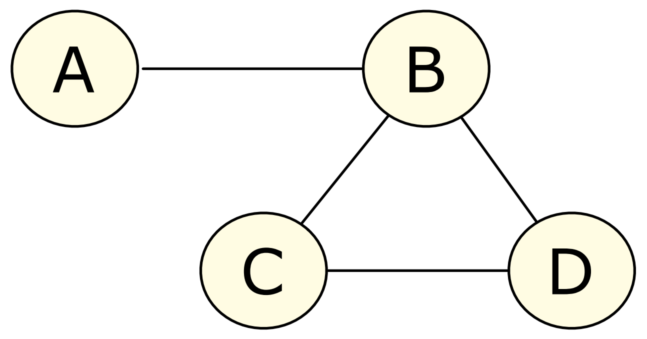
- We now have the computational resources to perform network analysis on large-scale data...

facebook
FROM FACEBOOK
December 2010

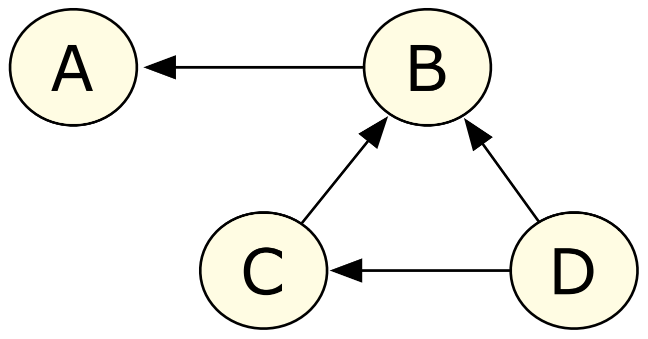
http://www.facebook.com/note.php?note_id=469716398919

Basic Concepts

- Graph: a way of representing the relationships among a collection of objects.
- Consists of a set of objects, called nodes, with certain pairs of these objects connected by links called edges.



Undirected Graph



Directed Graph

- Two nodes are neighbours if they are connected by an edge.
- Degree of a node is the number of edges ending at that node.
- For a directed graph, the in-degree and out-degree of a node refer to numbers of edges incoming to or outgoing from the node.

NetworkX - Creating Graphs

```
>>> import networkx
```

Import library

```
>>> g = networkx.Graph()
```

Create new undirected graph

```
>>> g.add_node("John")
>>> g.add_node("Maria")
>>> g.add_node("Alex")
>>> g.add_edge("John", "Alex")
>>> g.add_edge("Maria", "Alex")
```

Add new nodes with unique IDs.

Add new edges referencing associated node IDs.

```
>>> print g.number_of_nodes()
3
>>> print g.number_of_edges()
2
>>> print g.nodes()
['John', 'Alex', 'Maria']
>>> print g.edges()
[('John', 'Alex'), ('Alex', 'Maria')]
```

Print details of our newly-created graph.

```
>>> print g.degree("John")
1
>>> print g.degree()
{'John': 1, 'Alex': 2, 'Maria': 1}
```

Calculate degree of specific node, or map of degree for all nodes.

NetworkX - Directed Graphs

```
>>> g = networkx.DiGraph()
```

Create new directed graph

```
>>> g.add_edges_from([("A","B"), ("C","A")])
```

Edges can be added in batches.

```
>>> print g.in_degree(with_labels=True)
{'A': 1, 'C': 0, 'B': 1}
>>> print g.out_degree(with_labels=True)
{'A': 1, 'C': 1, 'B': 0}
```

Nodes can be added to the graph "on the fly".

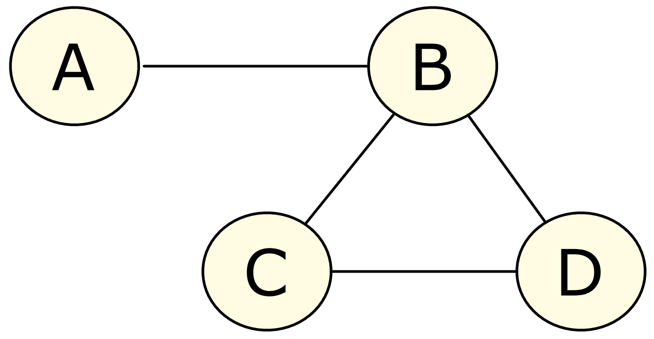
```
>>> print g.neighbors("A")
['B']
>>> print g.neighbors("B")
[]
```

```
>>> ug = g.to_undirected()
>>> print ug.neighbors("B")
['A']
```

Convert to an undirected graph

NetworkX - Loading Existing Graphs

- Library includes support for reading/writing graphs in a variety of file formats.



Edge List Format

```
a b
b c
b d
c d
```

Node pairs, one edge per line.

```
>>> g = networkx.read_edgelist("test.edges")
>>> print g.edges()
[('a', 'b'), ('c', 'b'), ('c', 'd'), ('b', 'd')]
```

Adjacency List Format

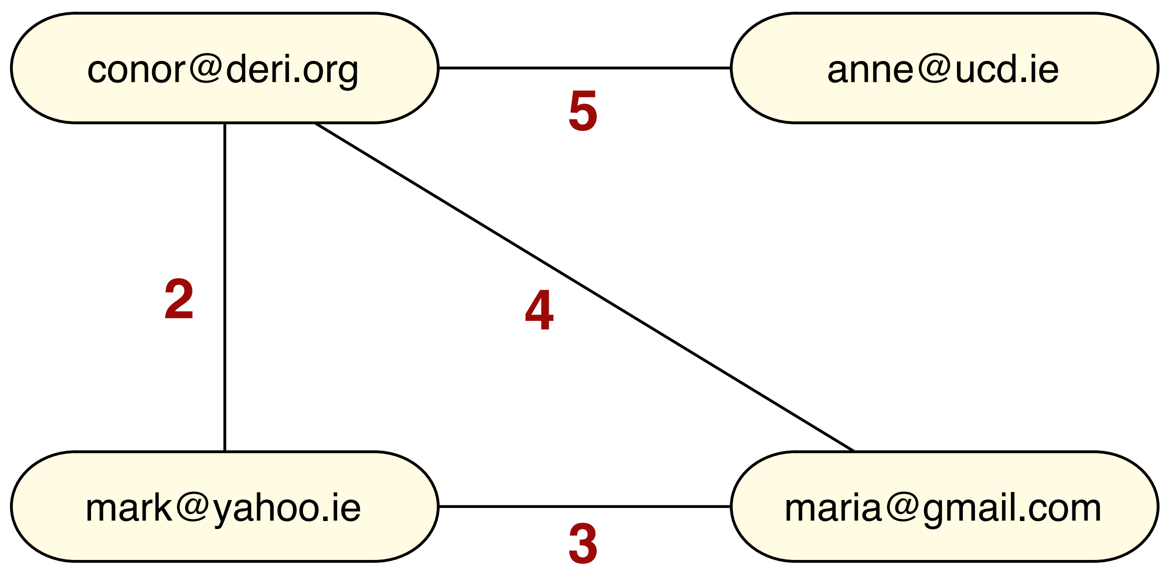
```
a b
b c d
c d
```

First label in line is the source node. Further labels in the line are considered target nodes.

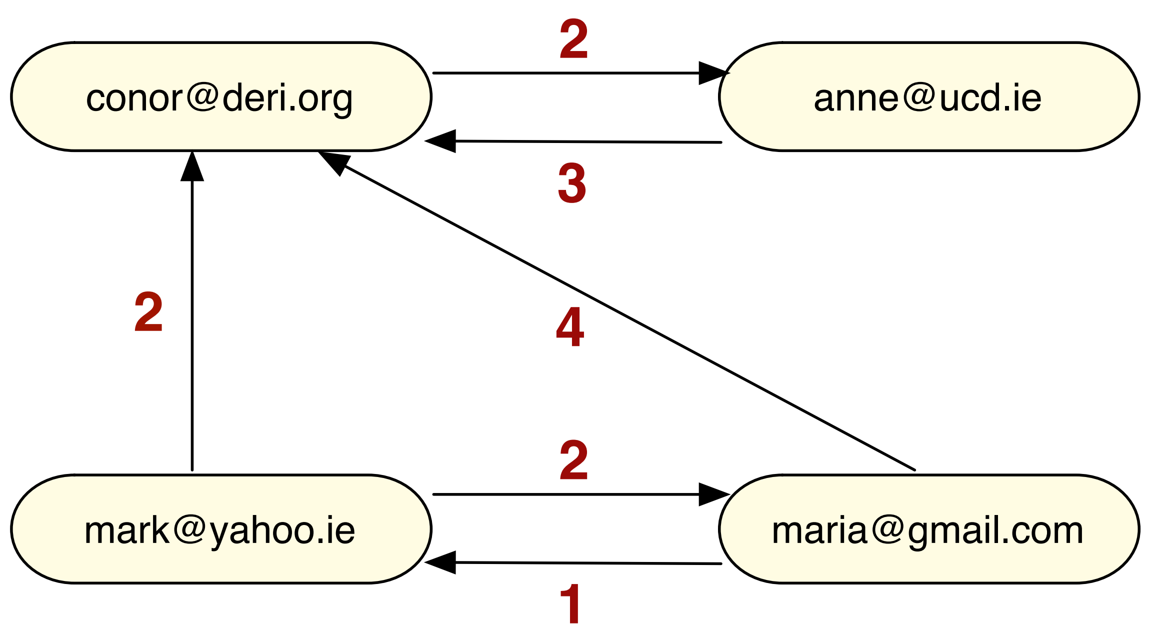
```
>>> g = networkx.read_adjlist("test_adj.txt")
>>> print edges()
[('a', 'b'), ('c', 'b'), ('c', 'd'), ('b', 'd')]
```

Weighted Graphs

- Weighted graph: numeric value is associated with each edge.
- Edge weights may represent a concept such as similarity, distance, or connection cost.



Undirected weighted graph



Directed weighted graph

NetworkX - Weighted Graphs

```
g = networkx.Graph()
```

```
g.add_edge("conor@deri.org", "anne@ucd.ie", weight=5)
g.add_edge("conor@deri.org", "mark@yahoo.ie", weight=2)
g.add_edge("conor@deri.org", "maria@gmail.com", weight=4)
g.add_edge("mark@yahoo.ie", "maria@gmail.com", weight=3)
```

Add weighted edges to graph.

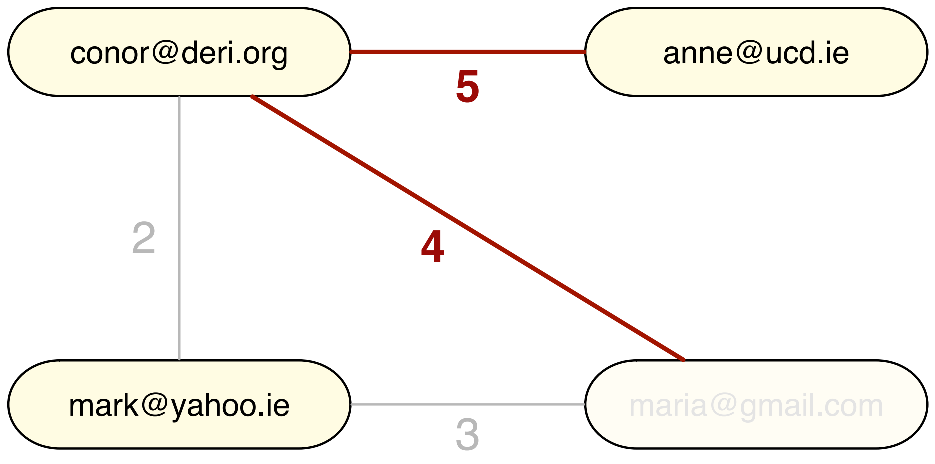
Note: nodes can be added to the graph "on the fly"

Select the subset of "strongly weighted" edges above a threshold...

```
estrong = [(u,v) for (u,v,d) in g.edges(data=True) if d["weight"] > 3]
```

```
>>> print estrong
[('conor@deri.org', 'anne@ucd.ie'), ('conor@deri.org', 'maria@gmail.com')]
```

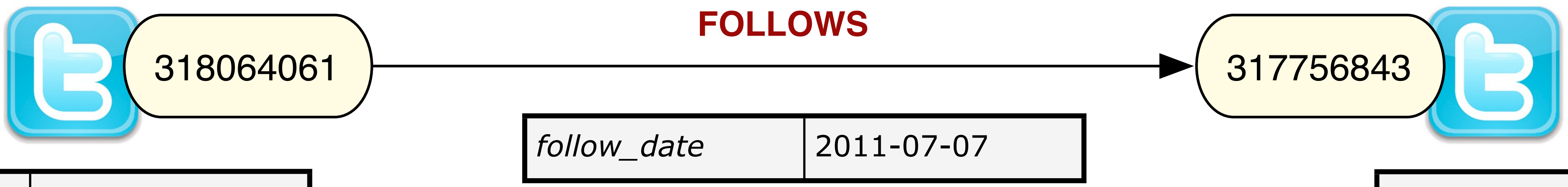
```
>>> print g.degree("conor@deri.org", weighted=False)
3
>>> print g.degree("conor@deri.org", weighted=True)
11
```



Weighted degree given by sum of edge weights.

Attributed Graphs

- Additional attribute data, relating to nodes and/or edges, is often available to compliment network data.



follow_date	2011-07-07

screen_name	peter78
location	Galway
time_zone	GMT
verified	FALSE

screen_name	mark763
location	London
time_zone	GMT
verified	FALSE

Create new nodes with attribute values

```
g.add_node("318064061", screen_name="peter78", location="Galway", time_zone="GMT")
g.add_node("317756843", screen_name="mark763", location="London", time_zone="GMT")
```

Add/modify attribute values for existing nodes

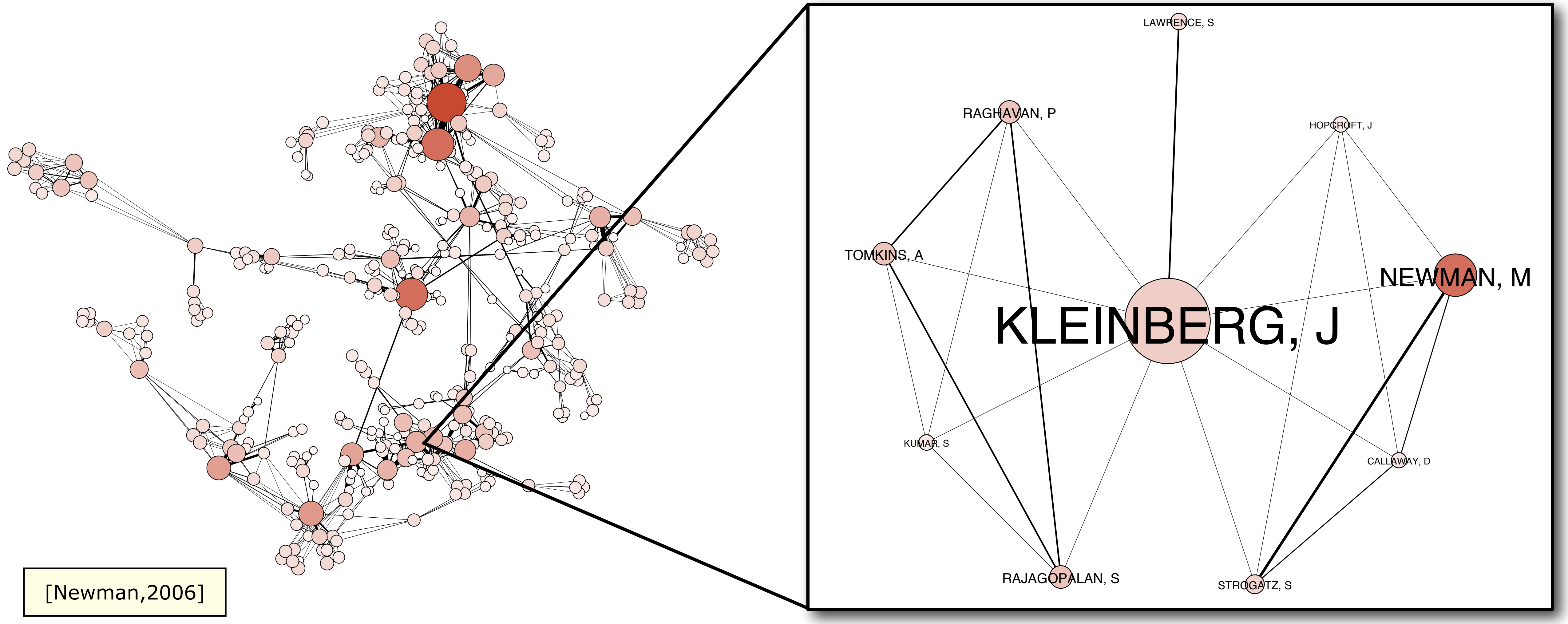
```
g.node["318064061"]["verified"] = False
g.node["317756843"]["verified"] = False
```

Create new edge with attribute values

```
g.add_edge("318064061", "317756843", follow_date=datetime.datetime.now())
```

Ego Networks

- Ego-centric methods really focus on the individual, rather than on network as a whole.
- By collecting information on the connections among the modes connected to a focal ego, we can build a picture of the **local** networks of the individual.



[Newman,2006]

NetworkX - Ego Networks

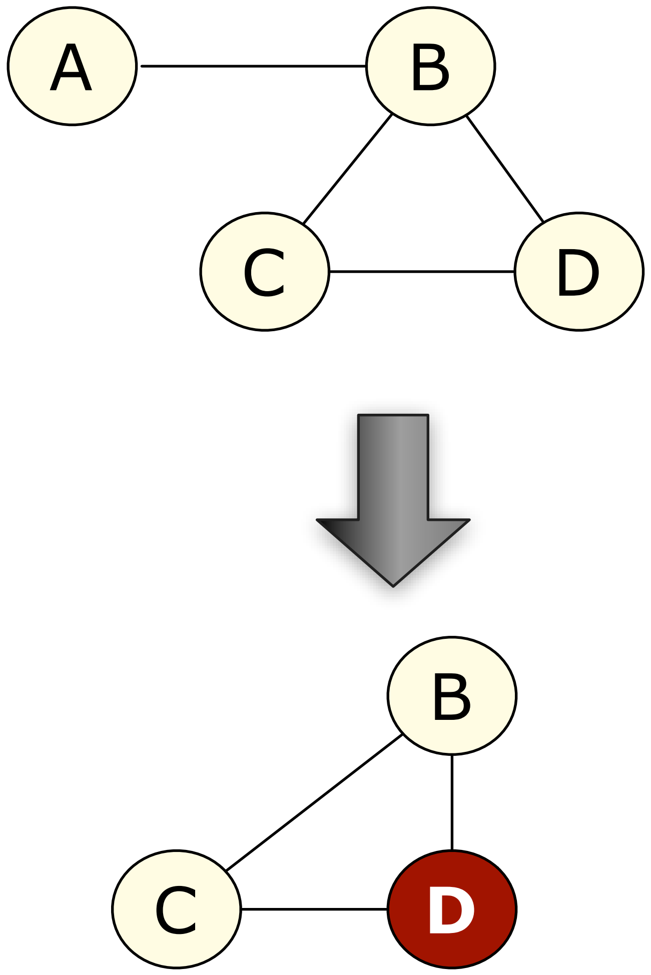
- We can readily construct an ego network subgraph from a global graph in NetworkX.

```
>>> g = networkx.read_adjlist("test.adj")
```

```
>>> ego = "d"
```

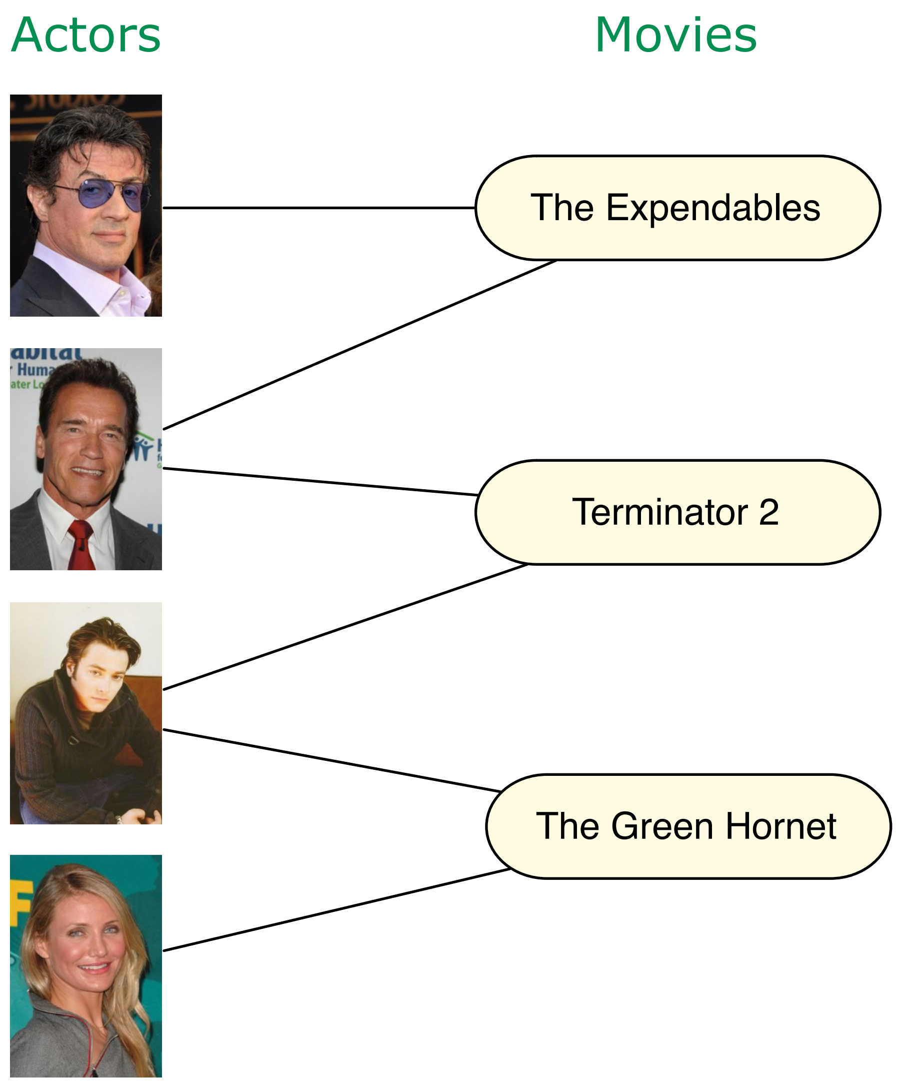
```
>>> nodes = set([ego])
>>> nodes.update(g.neighbors(ego))
>>> egonet = g.subgraph(nodes)
```

```
>>> print egonet.nodes()
['c', 'b', 'd']
>>> print egonet.edges()
[('c', 'b'), ('c', 'd'), ('b', 'd')]
```

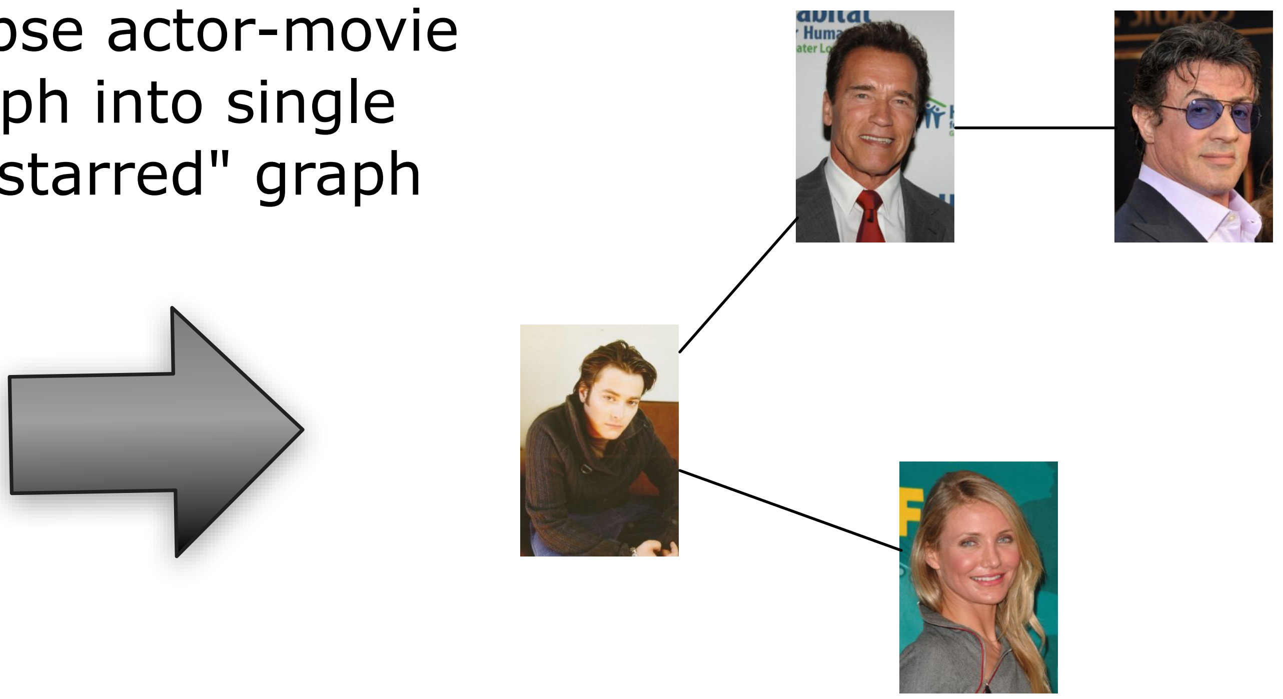


Bipartite Graphs

- In a bipartite graph the nodes can be divided into two disjoint sets so that no pair of nodes in the same set share an edge.



Collapse actor-movie graph into single "co-starred" graph



NetworkX - Bipartite Graphs

- NetworkX does not have a custom bipartite graph class.

➡ A standard graph can be used to represent a bipartite graph.

```
import networkx
from networkx.algorithms import bipartite
```

Import package for handling bipartite graphs

```
g = networkx.Graph()
```

Create standard graph, and add edges.

```
g.add_edges_from([("Stallone","Expendables"), ("Schwarzenegger","Expendables")])
g.add_edges_from([("Schwarzenegger","Terminator 2"), ("Furlong","Terminator 2")])
g.add_edges_from([("Furlong","Green Hornet"), ("Diaz","Green Hornet")])
```

```
>>> print bipartite.is_bipartite(g)
True
>>> print bipartite.bipartite_sets(g)
(set(['Stallone', 'Diaz', 'Schwarzenegger', 'Furlong']),
set(['Terminator 2', 'Green Hornet', 'Expendables']))
```

Verify our graph is bipartite, with two disjoint node sets.

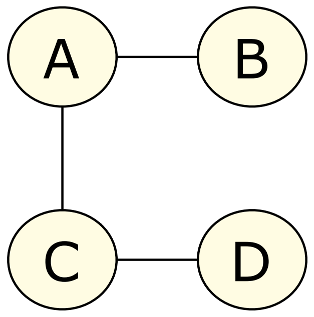
```
>>> g.add_edge("Schwarzenegger","Stallone")
>>> print bipartite.is_bipartite(g)
False
```

Graph is no longer bipartite!

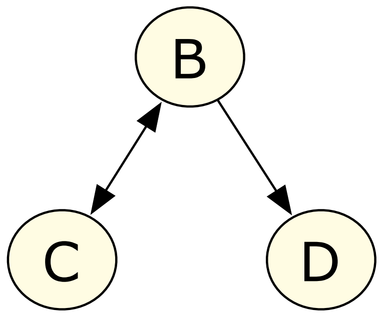
Multi-Relational Networks

- In many SNA applications there will be multiple kinds of relations between nodes. Nodes may be closely-linked in one relational network, but distant in another.

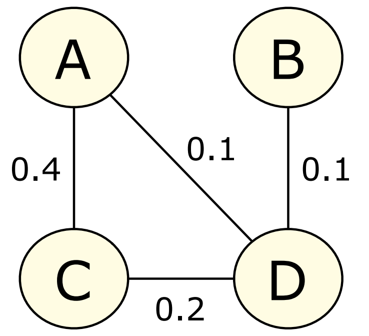
Scientific Research Network



Co-authorship Graph

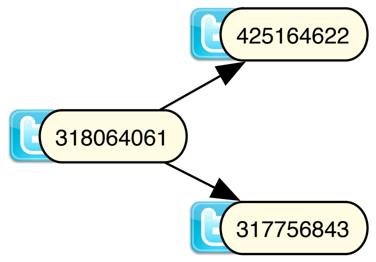


Citation Graph

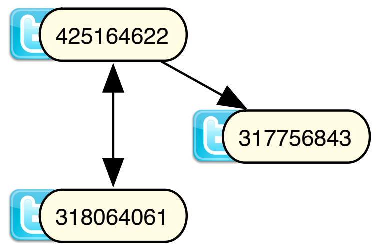


Content Similarity

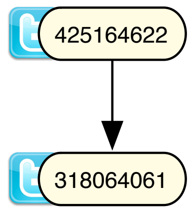
Microblogging Network



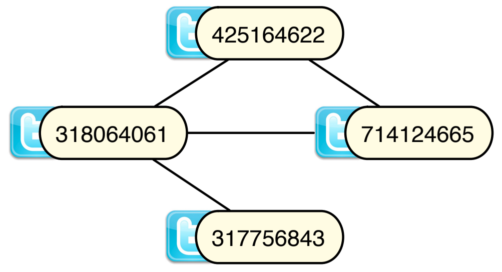
Follower Graph



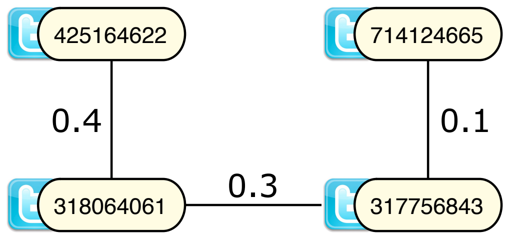
Reply-To Graph



Mention Graph



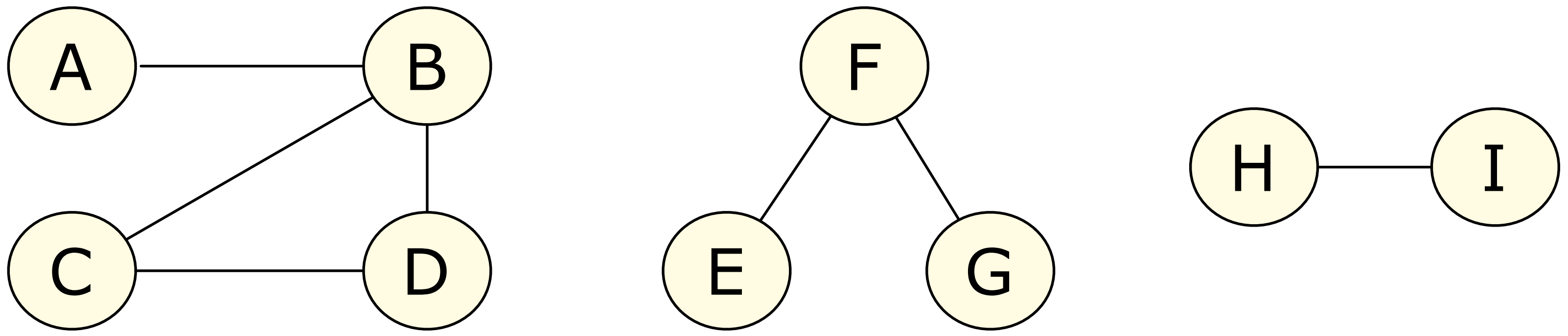
Co-Listed Graph



Content Similarity

Graph Connectivity - Components

- A graph is connected if there is a path between every pair of nodes in the graph.
- A connected component is a subset of the nodes where:
  1. A path exists between every pair in the subset.
  2. The subset is not part of a larger set with the above property.



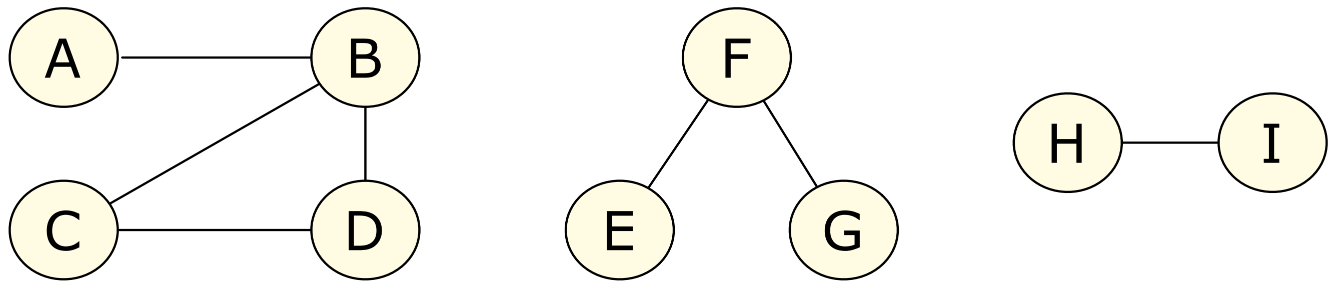
3 connected components

- In many empirical social networks a larger proportion of all nodes will belong to a single giant component.

NetworkX - Graph Connectivity

```
g = networkx.Graph()
g.add_edges_from([("a","b"),("b","c"),("b","d"),("c","d")])
g.add_edges_from([("e","f"),("f","g"),("h","i")])
```

Build undirected graph.



```
>>> print networkx.is_connected(g)
False
>>> print networkx.number_connected_components(g)
3
```

Is the graph just a single component?

If not, how many components are there?

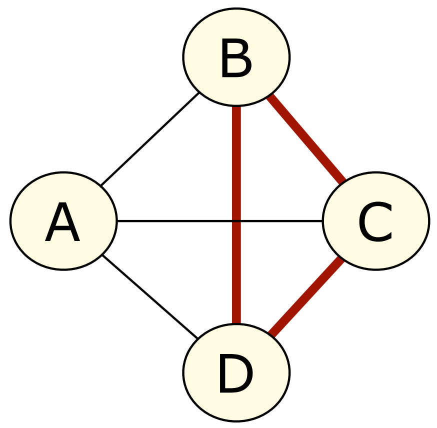
```
>>> comps = networkx.connected_component_subgraphs(g)
>>> print comps[0].nodes()
['a', 'c', 'b', 'd']
>>> print comps[1].nodes()
['e', 'g', 'f']
>>> print comps[2].nodes()
['i', 'h']
```

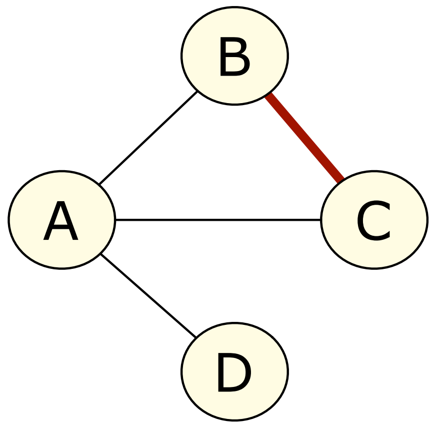
Find list of all connected components.

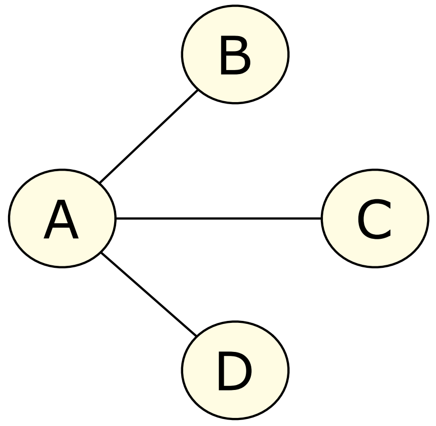
Each component is a subgraph with its own set of nodes and edges.

Clustering Coefficient

- The neighbourhood of a node is set of nodes connected to it by an edge, not including itself.
- The clustering coefficient of a node is the fraction of pairs of its neighbours that have edges between one another.





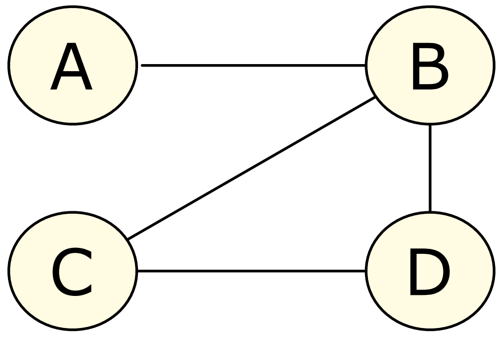


Node A: $CC = \frac{3}{3}$ $CC = \frac{1}{3}$ $CC = \frac{0}{3}$

- Locally indicates how concentrated the neighbourhood of a node is, globally indicates level of clustering in a graph.
- Global score is average over all nodes: $\bar{CC} = \frac{1}{n}\sum_{i=1}^{n} CC(v_i)$

NetworkX - Clustering Coefficient

```
g = networkx.Graph()
g.add_edges_from([("a","b"),("b","c"),("b","d"),("c","d")])
```



```
>>> print networkx.neighbors(g, "b")
['a', 'c', 'd']
```

Get list of neighbours for a specific node.

```
>>> print networkx.clustering(g, "b")
0.333333333333
```

Calculate coefficient for specific node.

```
>>> print networkx.clustering(g, with_labels=True)
{'a': 0.0, 'c': 1.0, 'b': 0.33333333333333331, 'd': 1.0}
```

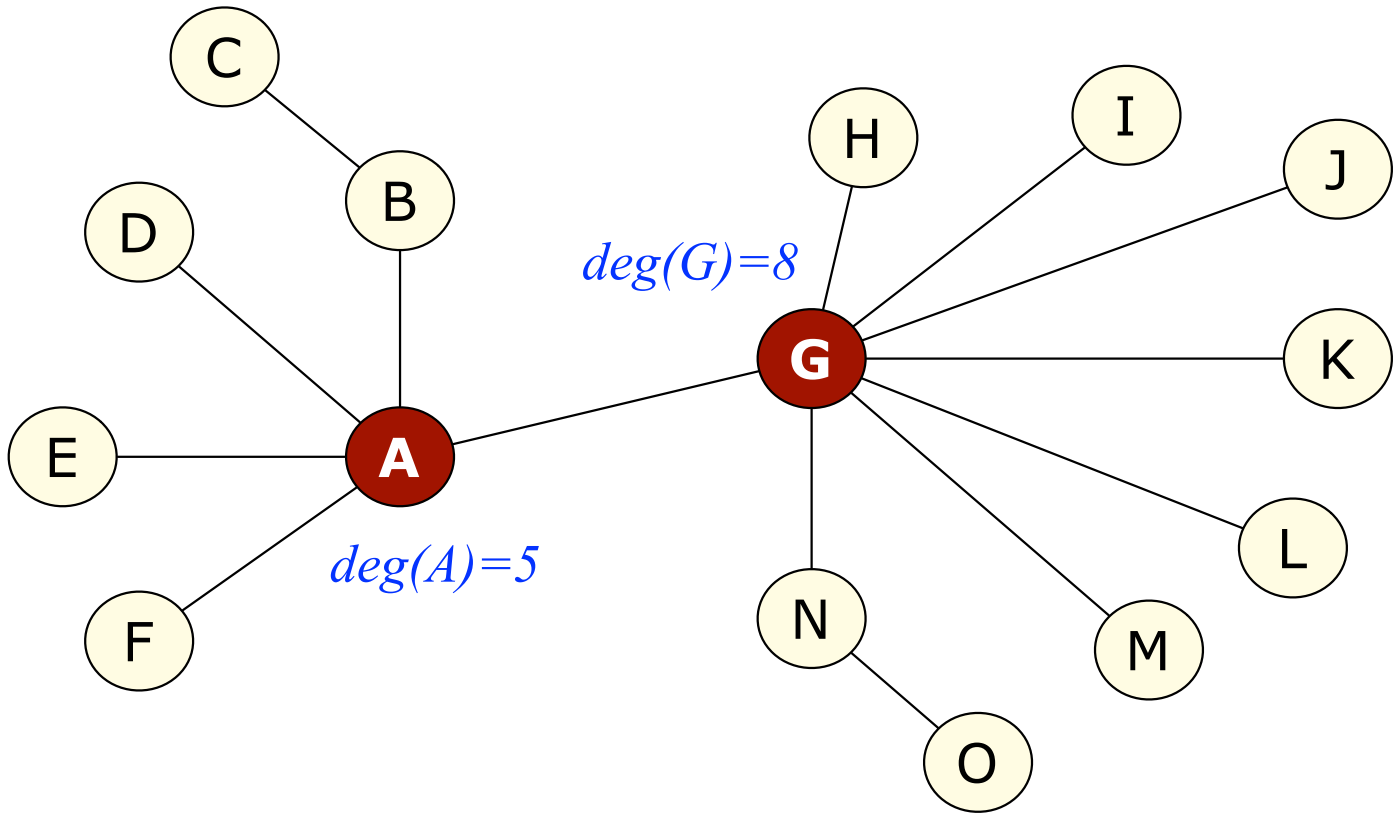
Build a map of coefficients for all nodes.

```
>>> ccs = networkx.clustering(g)
>>> print ccs
[0.0, 1.0, 0.33333333333333331, 1.0]
>>> print sum(ccs)/len(ccs)
0.583333333333
```

Calculate global clustering coefficient.

Measures of Centrality

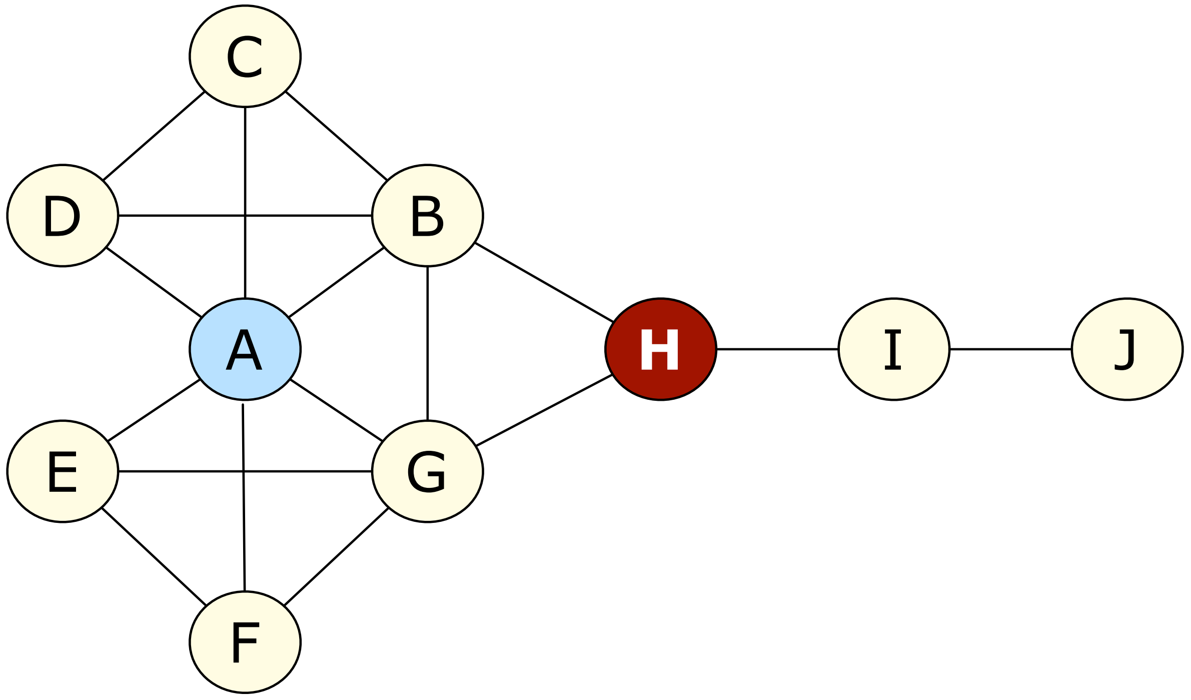
- A variety of different measures exist to measure the importance, popularity, or social capital of a node in a social network.
- Degree centrality focuses on individual nodes - it simply counts the number of edges that a node has.
- Hub nodes with high degree usually play an important role in a network. For directed networks, in-degree is often used as a proxy for popularity.



Betweenness Centrality

- A path in a graph is a sequence of edges joining one node to another. The path length is the number of edges.
- Often want to find the shortest path between two nodes.
- A graph's diameter is the longest shortest path over all pairs of nodes.

- Nodes that occur on many shortest paths between other nodes in the graph have a high betweenness centrality score.



Node "A" has high degree centrality than "B", as "B" has few direct connections.

Node "H" has higher betweenness centrality, as "H" plays a broker role in the network.

Eigenvector Centrality

- The eigenvector centrality of a node proportional to the sum of the centrality scores of its neighbours.

➡ A node is important if it connected to other important nodes.

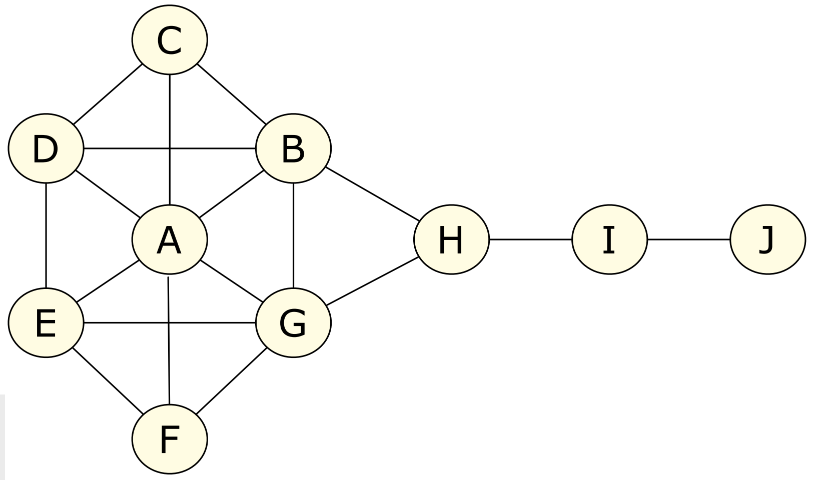
➡ A node with a small number of influential contacts may outrank one with a larger number of mediocre contacts.

- Computation:
  1. Calculate the eigendecomposition of the pairwise adjacency matrix of the graph.
  2. Select the eigenvector associated with largest eigenvalue.
  3. Element *i* in the eigenvector gives the centrality of the *i*-th node.

NetworkX - Measures of Centrality

```
import networkx
from operator import itemgetter
```

```
g = networkx.read_adjlist("centrality.edges")
```



```
dc = networkx.degree_centrality(g)
print sorted(dc.items(), key=itemgetter(1), reverse=True)
```

```
[('a', 0.66666666666666663), ('b', 0.55555555555555558), ('g', 0.55555555555555558), ('c', 0.33333333333333331),
('e', 0.33333333333333331), ('d', 0.33333333333333331), ('f', 0.33333333333333331), ('h', 0.33333333333333331),
('i', 0.22222222222222221), ('j', 0.1111111111111111)]
```

```
bc = networkx.betweenness_centrality(g)
print sorted(bc.items(), key=itemgetter(1), reverse=True)
```

```
[('h', 0.38888888888888884), ('b', 0.2361111111111111), ('g', 0.2361111111111111), ('i', 0.22222222222222221),
('a', 0.16666666666666666), ('c', 0.0), ('e', 0.0), ('d', 0.0), ('f', 0.0), ('j', 0.0)]
```

```
bc = networkx.eigenvector_centrality(g)
print sorted(bc.items(), key=itemgetter(1), reverse=True)
```

```
[('a', 0.17589997921479006), ('b', 0.14995290497083508), ('g', 0.14995290497083508), ('c', 0.10520440827586457),
('e', 0.10520440827586457), ('d', 0.10520440827586457), ('f', 0.10520440827586457), ('h', 0.078145778134411939),
('i', 0.020280613919932109), ('j', 0.0049501856857375875)]
```

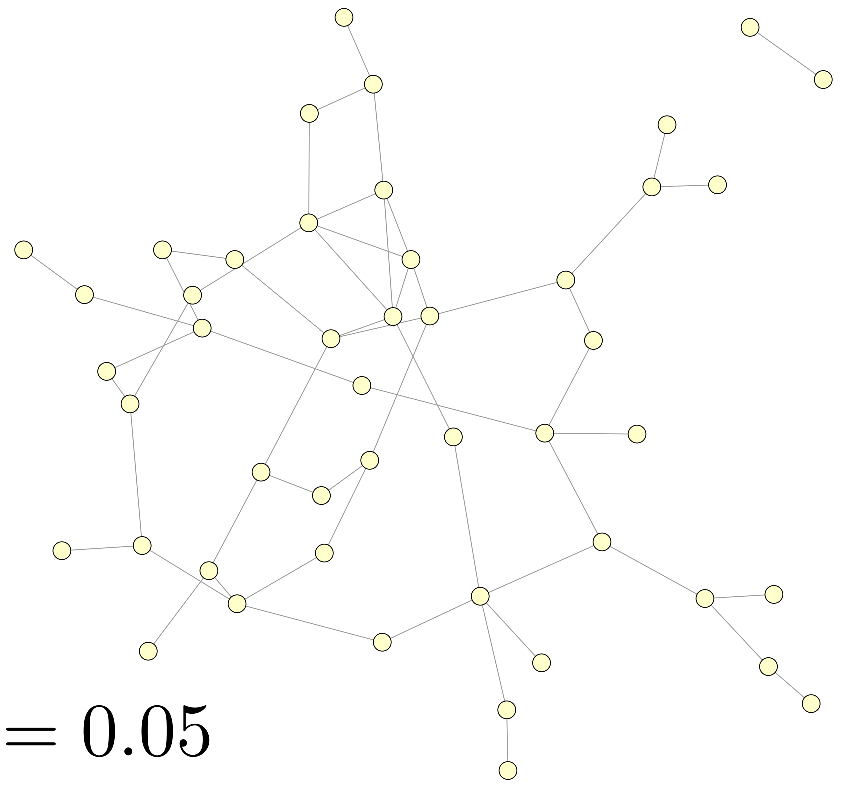
Random Networks

- Erdős–Rényi random graph model:
  - Start with a collection of *n* disconnected nodes.
  - Create an edge between each pair of nodes with a probability *p*, independently of every other edge.

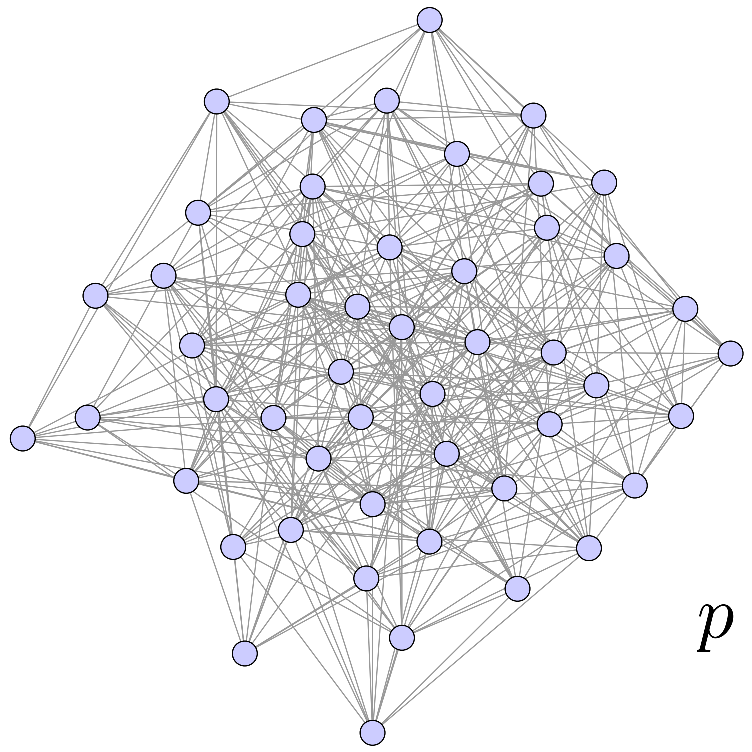
```
g1 = networkx.erdos_renyi_graph(50, 0.05)
```

```
g2 = networkx.erdos_renyi_graph(50, 0.3)
```

Specify number of nodes to create, and connection probability *p*.



$p = 0.05$



$p = 0.3$

Small World Networks

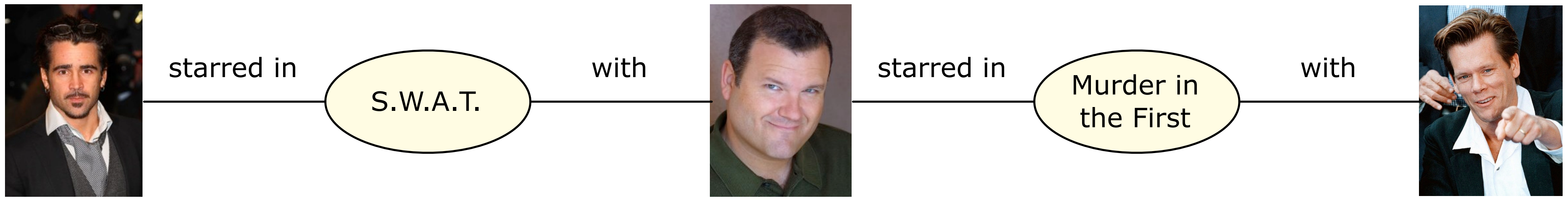
Milgram's Small World Experiment:

- Route a package to a stockbroker in Boston by sending them to random people in Nebraska and requesting them to forward to someone who might know the stockbroker.

➡ Although most nodes are not directly connected, each node can be reached from another via a relatively small number of hops.

Six Degrees of Kevin Bacon

- Examine the actor-actor "co-starred" graph from IMDB.
- The Bacon Number of an actor is the number of degrees of separation he/she has from Bacon, via the shortest path.



$\Rightarrow$ Bacon Number $= 2$

http://oracleofbacon.org

Small World Networks

- Take a connected graph with a high diameter, randomly add a small number of edges, then the diameter tends to drop drastically.
- Small-world network has many local links and few long range "shortcuts".

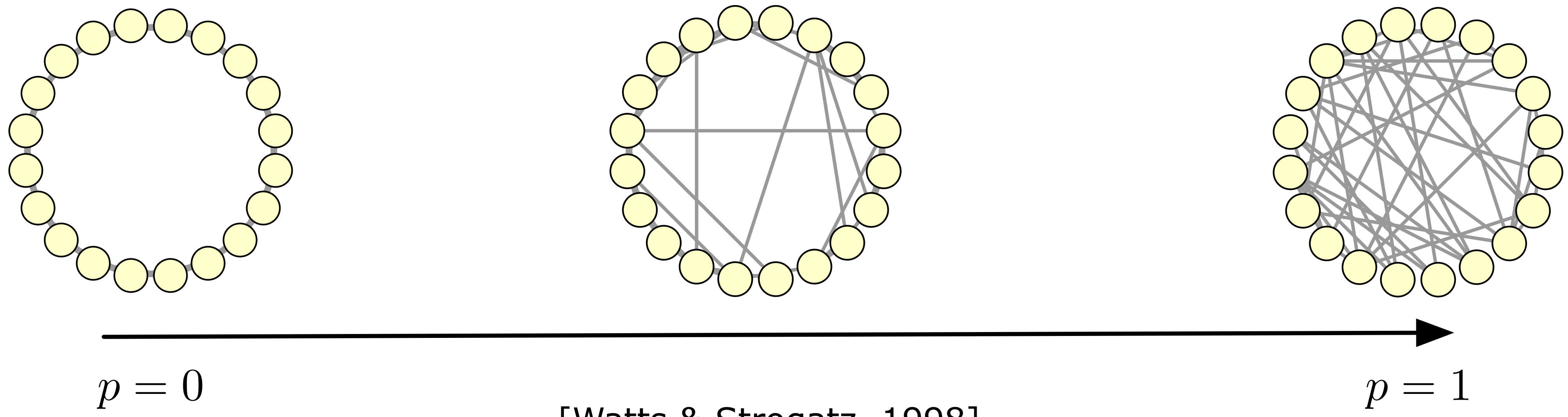
Typical properties:

- High clustering coefficient.
- Short average path length.
- Over-abundance of hub nodes.

[Watts & Strogatz, 1998]

Generating Small World Networks:

1. Create ring of *n* nodes, each connected to its *k* nearest neighbours.
2. With probability *p*, rewire each edge to an existing destination node.



[Watts & Strogatz, 1998]

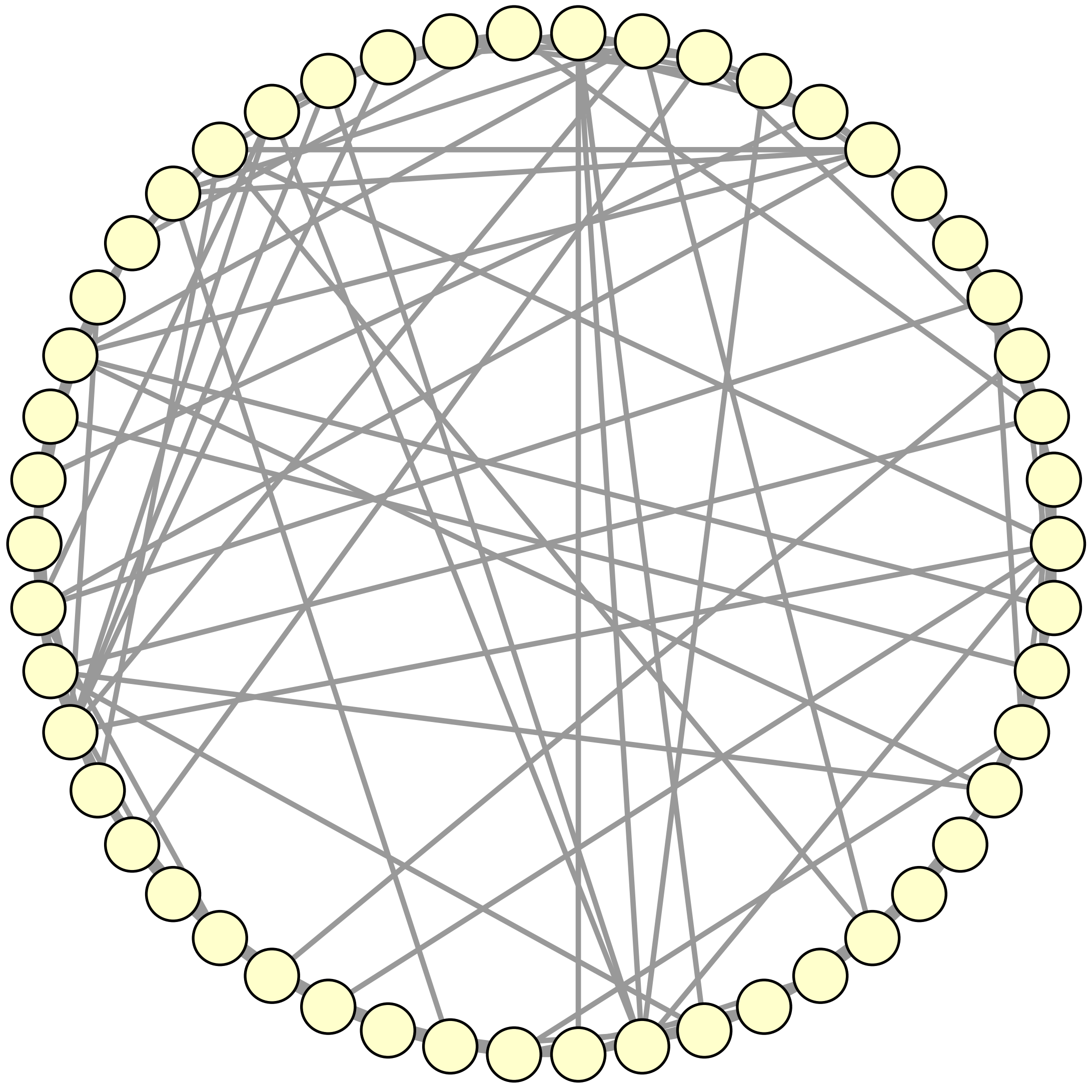
NetworkX - Small World Networks

- NetworkX includes functions to generate graphs according to a variety of well-known models:

  http://networkx.lanl.gov/reference/generators.html

```
n = 50
k = 6
p = 0.3
g = networkx.watts_strogatz_graph(n, k, p)
```

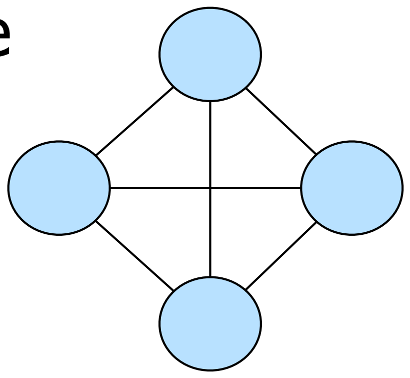
```
>>> networkx.average_shortest_path_length(g)
2.4506122448979597
```



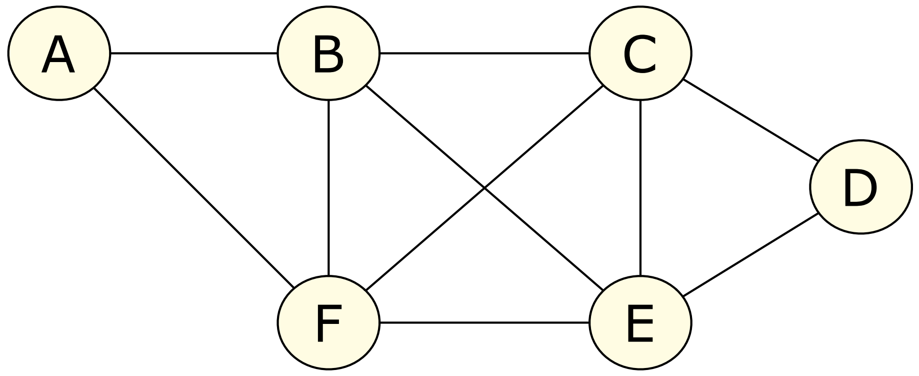
Community Finding

Cliques

- A clique is a social grouping where everyone knows everyone else (i.e. there is an edge between each pair of nodes).



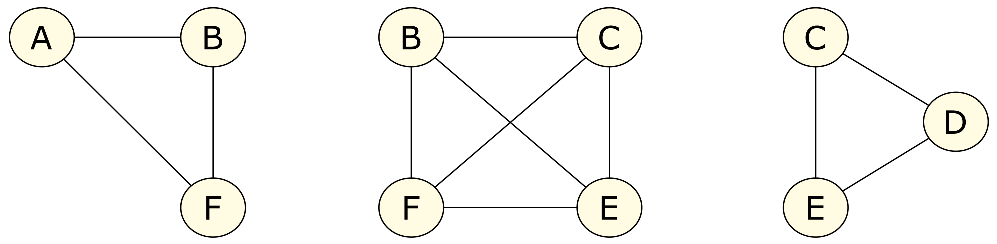
- A maximal clique is a clique that is not a subset of any other clique in the graph.
- A clique with size greater than or equal to that of every other clique in the graph is called a maximum clique.



Find all maximal cliques in the specified graph:

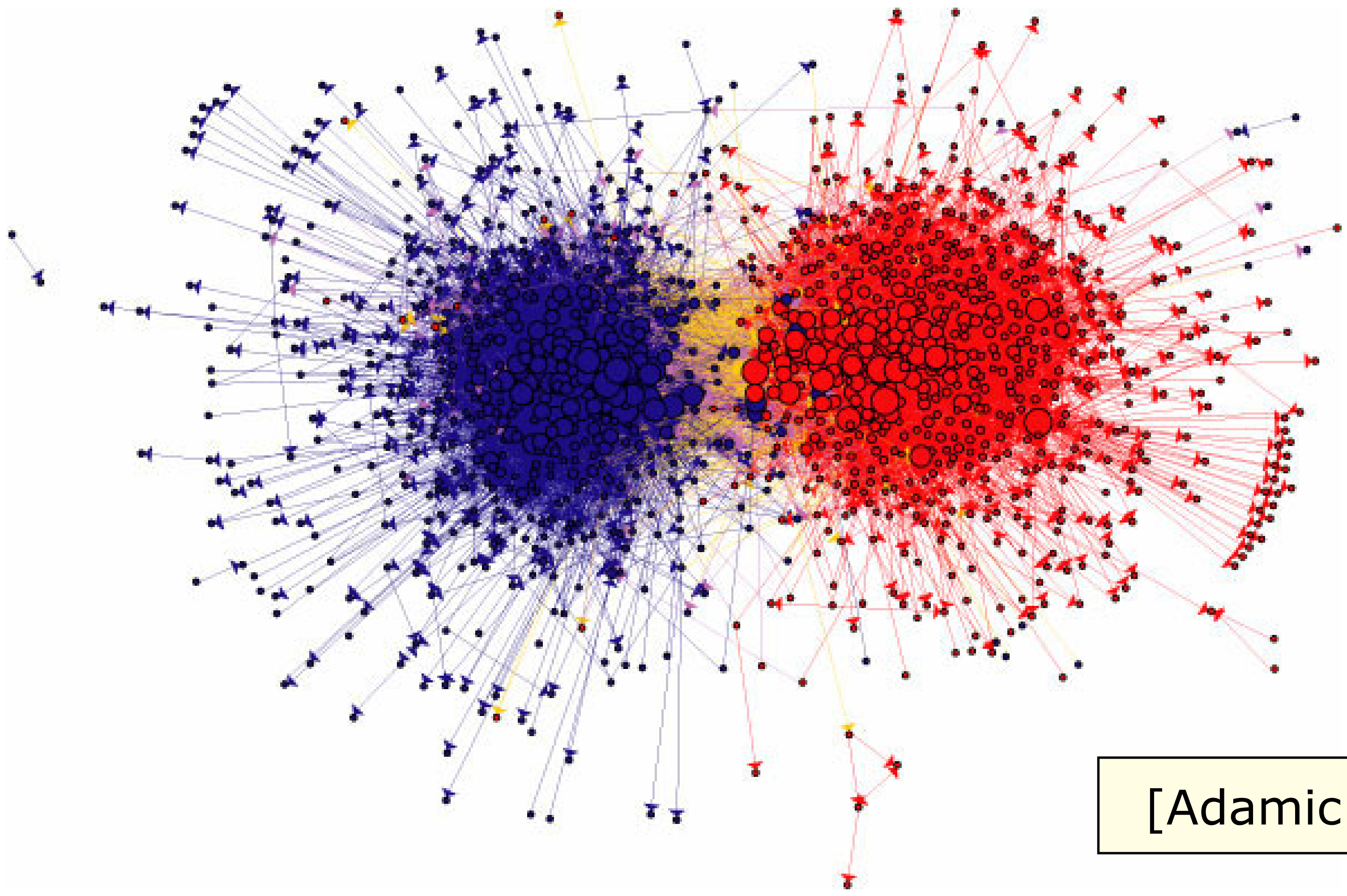
```
>>> cl = list( networkx.find_cliques(g) )
```

```
>>> print cl
[['a', 'b', 'f'], ['c', 'e', 'b', 'f'], ['c', 'e', 'd']]
```



Community Detection

- We will often be interested in identifying communities of nodes in a network...

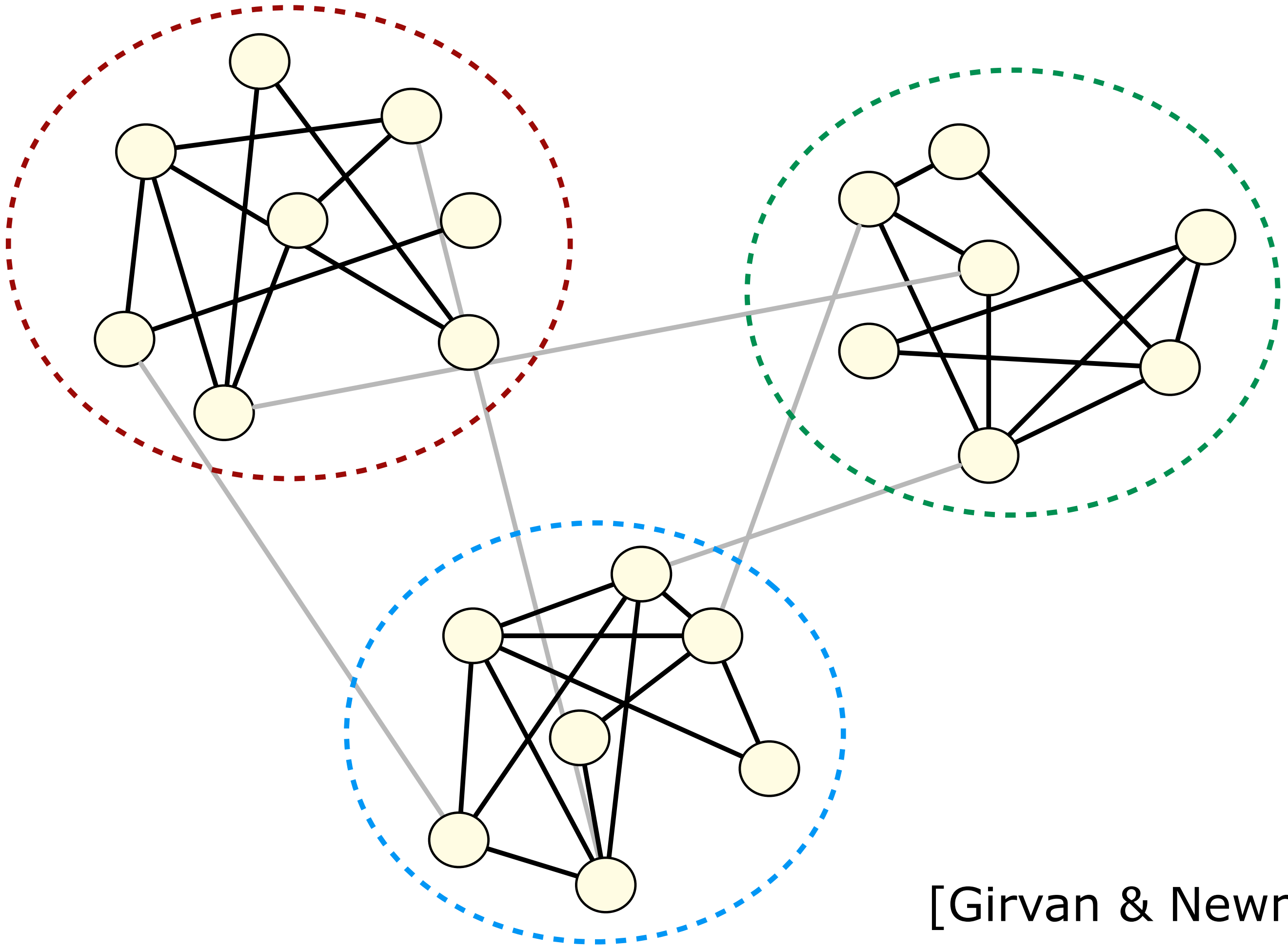


[Adamic & Glance,2005]

- Example: Two distinct communities of bloggers discussing 2004 US Presidential election.

Community Detection

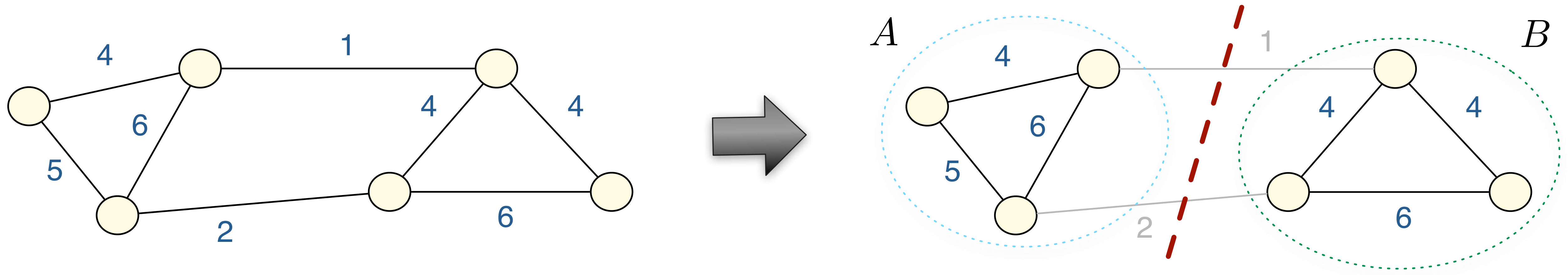
- A variety of definitions of community/cluster/module exist:
  - A group of nodes which share common properties and/or play a similar role within the graph [Fortunato, 2010].
  - A subset of nodes within which the node-node connections are dense, and the edges to nodes in other communities are less dense [Girvan & Newman, 2002].



[Girvan & Newman, 2002]

Graph Partitioning

- **Goal:** Divide the nodes in a graph into a user-specified number of disjoint groups to optimise a criterion related to number of edges cut.



- Min-cut simply involves minimising number (or weight) of edges cut by the partition.

$$cut(A, B) = 3$$

- Recent approaches use more sophisticated criteria (e.g. normalised cuts) and apply multi-level strategies to scale to large graphs.

  Graclus [Dhillon et al, 2007] http://www.cs.utexas.edu/users/dml/Software/graclus.html

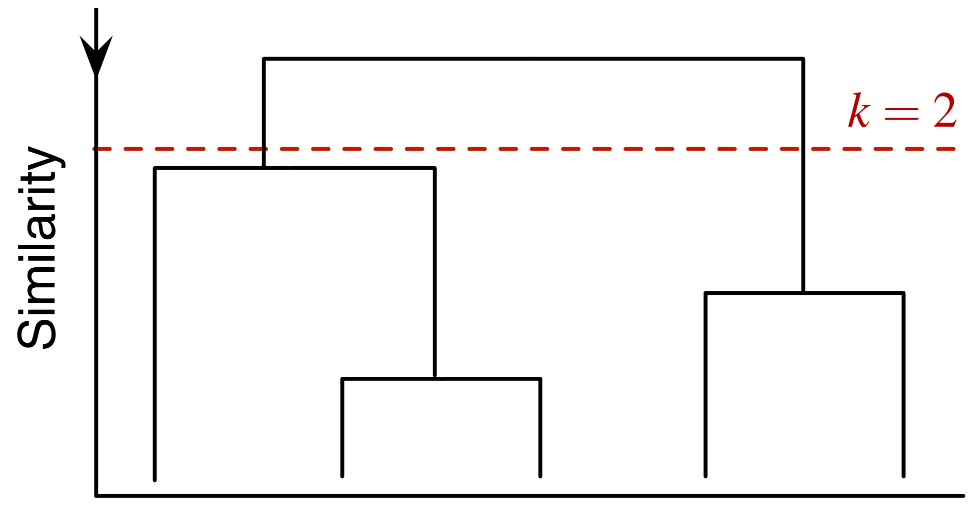
Issues: Requirement to pre-specify number of partitions, cut criteria often make strong assumptions about cluster structures.

Hierarchical Clustering

- Construct a tree of clusters to identify groups of nodes with high similarity according to some similarity measure.
- Two basic families of algorithm...
  1. Agglomerative: Begin with each node assigned to a singleton cluster. Apply a bottom-up strategy, merging the most similar pair of clusters at each level.
  2. Divisive: Begin with single cluster containing all nodes. Apply a top-down strategy, splitting a chosen cluster into two sub-clusters at each level.

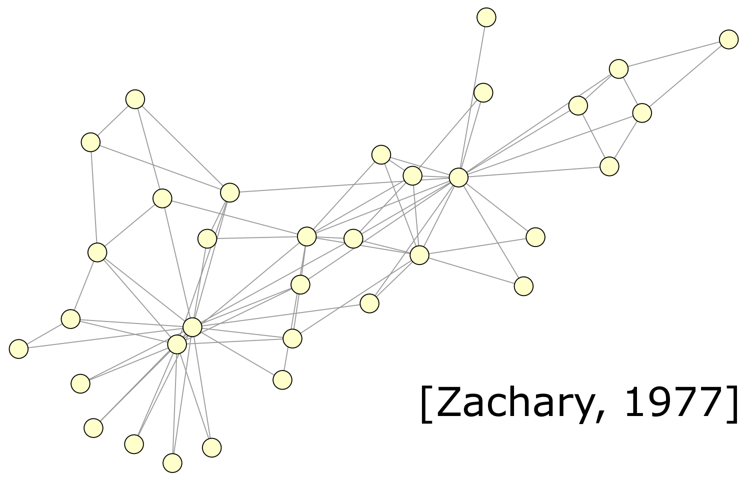
<u>Issues for Community Detection:</u>

- How do we choose among many different possible clusterings?
- Is there really a hierarchical structure in the graph?
- Often scales poorly to large graphs.



NetworkX - Hierarchical Clustering

- We can apply agglomerative clustering to a NetworkX graph by calling functions from the NumPy and SciPy numerical computing packages.



[Zachary, 1977]

```
import networkx
import numpy, matplotlib
from scipy.cluster import hierarchy
from scipy.spatial import distance
```

```
g = networkx.read_edgelist("karate.edgelist")
```

```
path_length=networkx.all_pairs_shortest_path_length(g)
n = len(g.nodes())
distances=numpy.zeros((n,n))
for u,p in path_length.iteritems():
   for v,d in p.iteritems():
      distances[int(u)-1][int(v)-1] = d
sd = distance.squareform(distances)
```

Build pairwise distance matrix based on shortest paths between nodes.

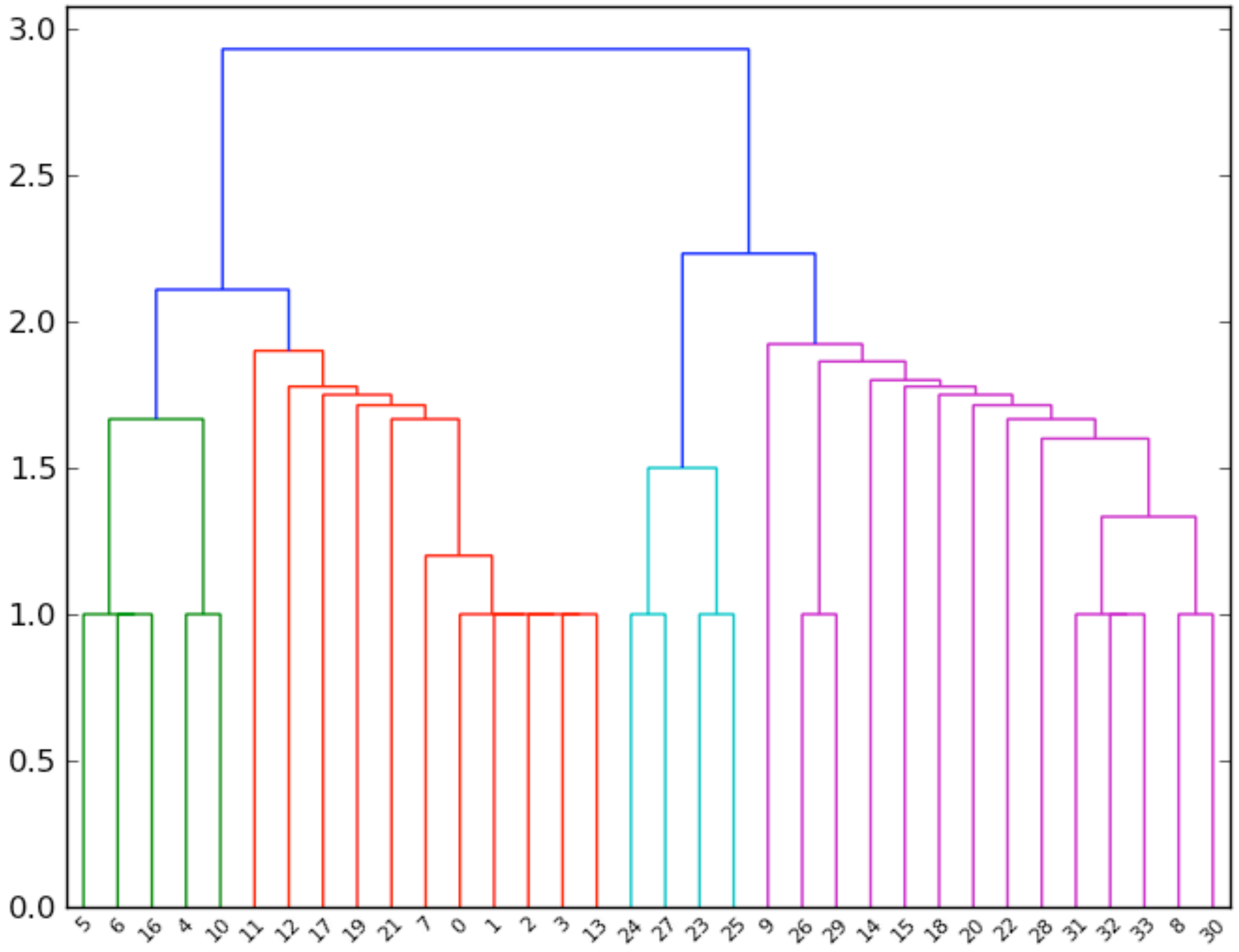
```
hier = hierarchy.average(sd)
```

Apply average-linkage agglomerative clustering.

NetworkX - Hierarchical Clustering

```
hierarchy.dendrogram(hier)
matplotlib.pylab.savefig("tree.png",format="png")
```

Build the dendrogram, then write image to disk.



Modularity Optimisation

- Newman & Girvan [2004] proposed measure of partition quality....
  - ➡ Random graph shouldn't have community structure.
  - ➡ Validate existence of communities by comparing actual edge density with expected edge density in random graph.

$$Q = (\text{number of edges within communities}) - (\text{expected number within communities})$$

- Apply agglomerative technique to iteratively merge groups of nodes to form larger communities such that modularity increases after merging.
- Recently efficient greedy approaches to modularity maximisation have been developed that scale to graphs with up to 10^9 edges.

Louvain Method [Blondel et al, 2008] http://findcommunities.googlepages.com

Issues for Community Detection:

- Total number of edges in graph controls the resolution at which communities are identified [Fortunato, 2010].
- Is it realistic/useful to assign nodes to only a single community?

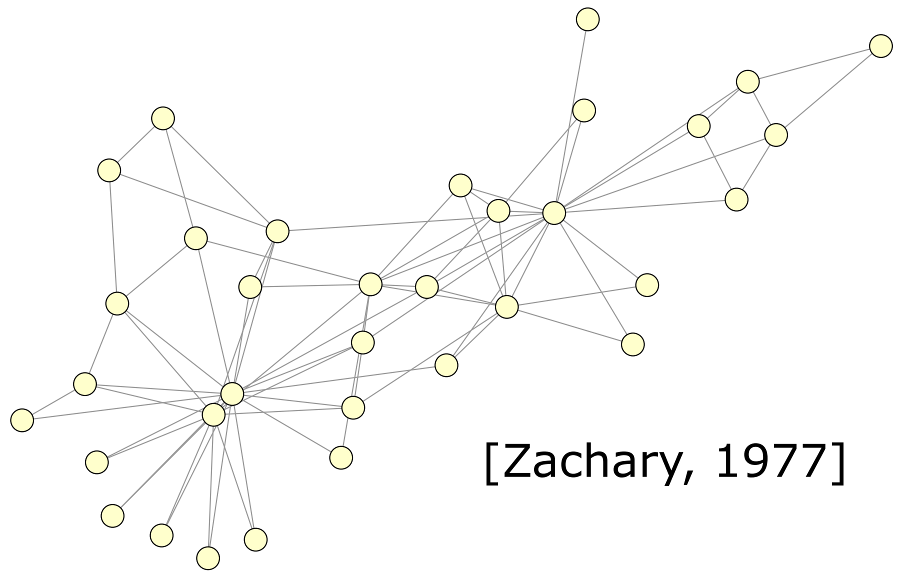
NetworkX - Modularity Optimisation

- Python implementation of the Louvain algorithm available: http://perso.crans.org/aynaud/communities/community.py

```
g = networkx.read_edgelist("karate.edges")
```

Apply Louvain algorithm to the graph

```
import community
partition = community.best_partition( g )
```


[Zachary, 1977]

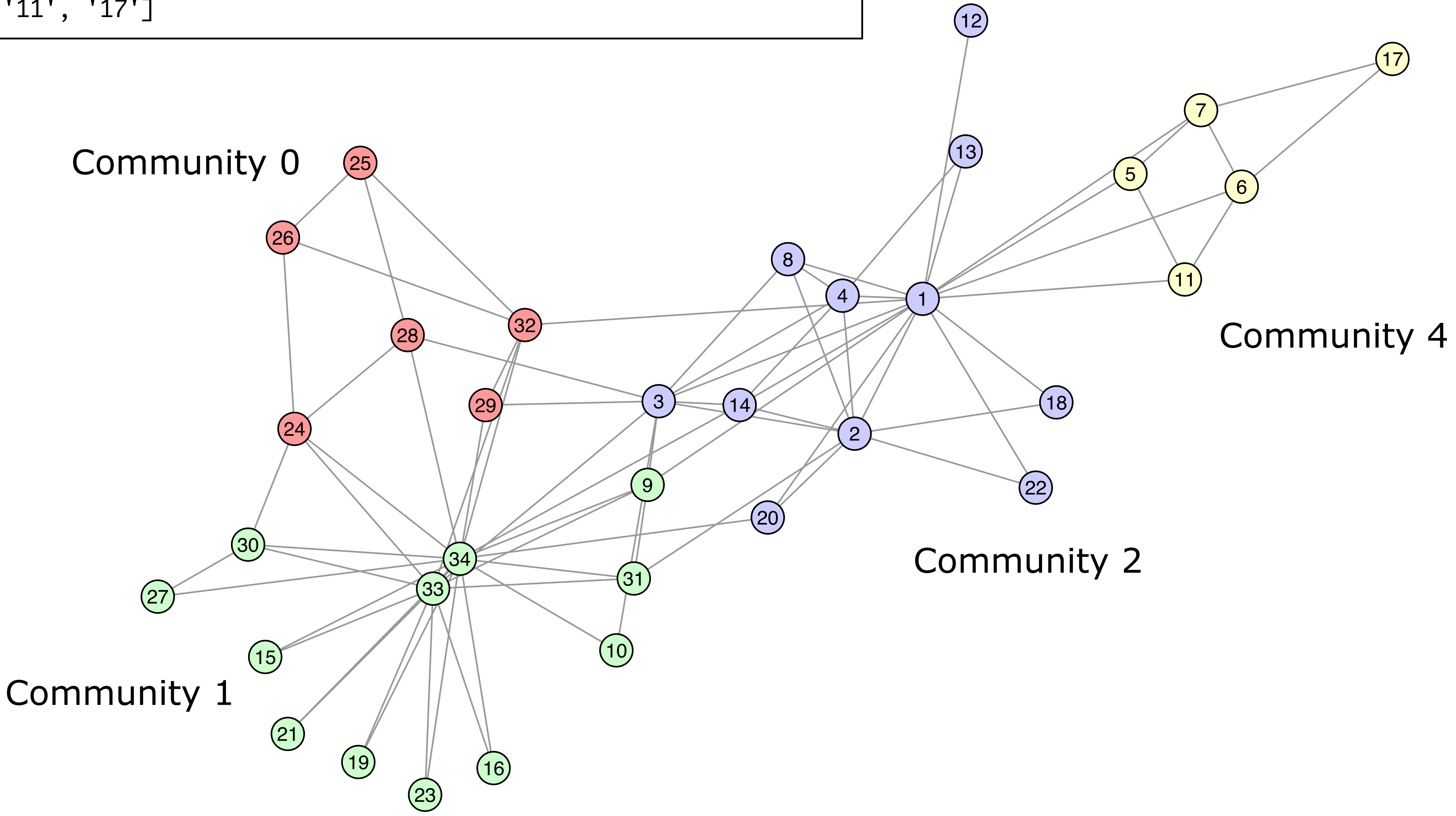
Print nodes assigned to each community in the partition

```
for i in set(partition.values()):
   print "Community", i
   members = list_nodes = [nodes for nodes in partition.keys() if partition[nodes] == i]
   print members
```

```
Community 0
['24', '25', '26', '28', '29', '32']
Community 1
['27', '21', '23', '9', '10', '15', '16', '33', '31', '30', '34', '19']
Community 2
['20', '22', '1', '3', '2', '4', '8', '13', '12', '14', '18']
Community 3
['5', '7', '6', '11', '17']
```

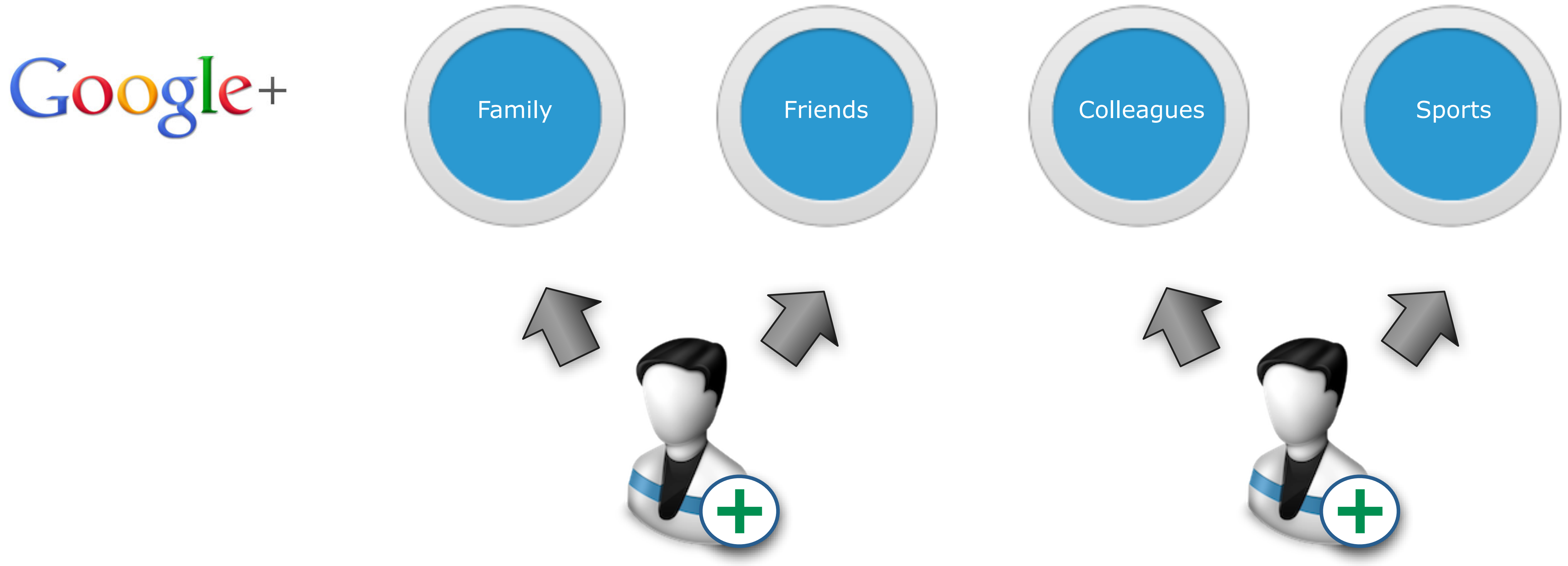
NetworkX - Modularity Optimisation

```
Community 0
['24', '25', '26', '28', '29', '32']
Community 1
['27', '21', '23', '9', '10', '15', '16', '33', '31', '30', '34', '19']
Community 2
['20', '22', '1', '3', '2', '4', '8', '13', '12', '14', '18']
Community 3
['5', '7', '6', '11', '17']
```



Overlapping v Non-Overlapping

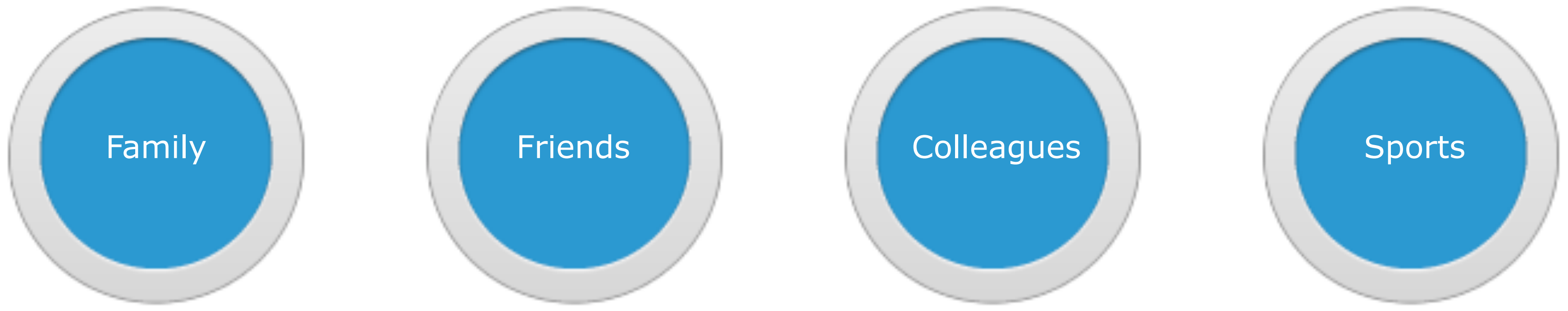
- Do disjoint non-overlapping communities make sense in empirical social networks?



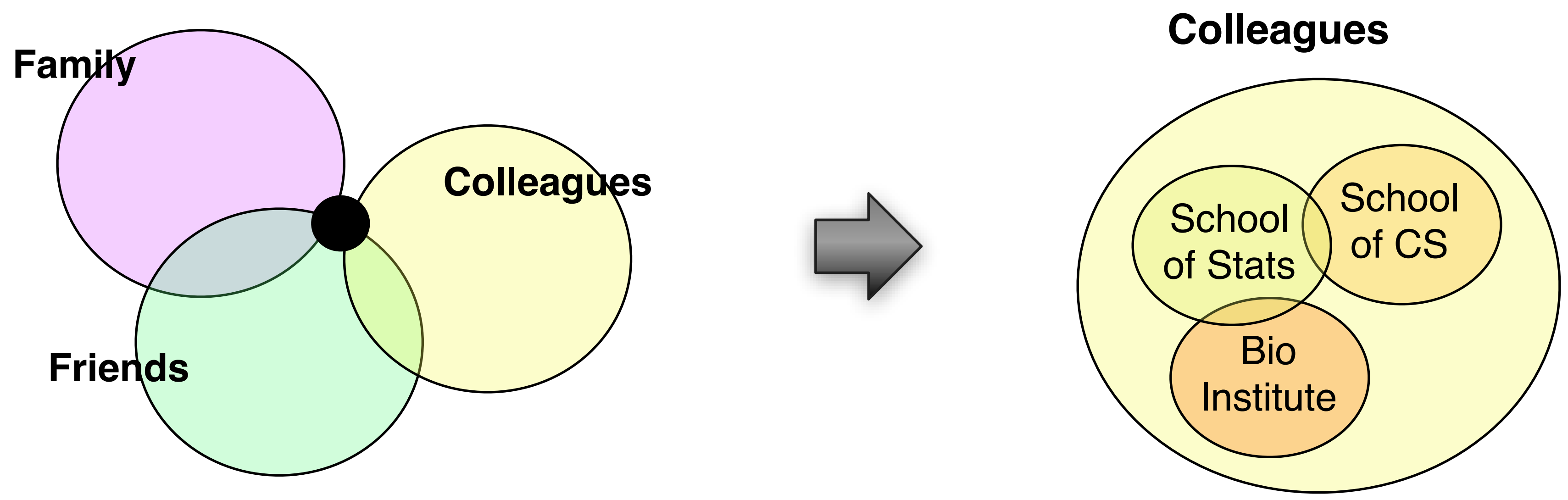
Overlapping v Non-Overlapping

- Do disjoint non-overlapping communities make sense in empirical social networks?

Google+



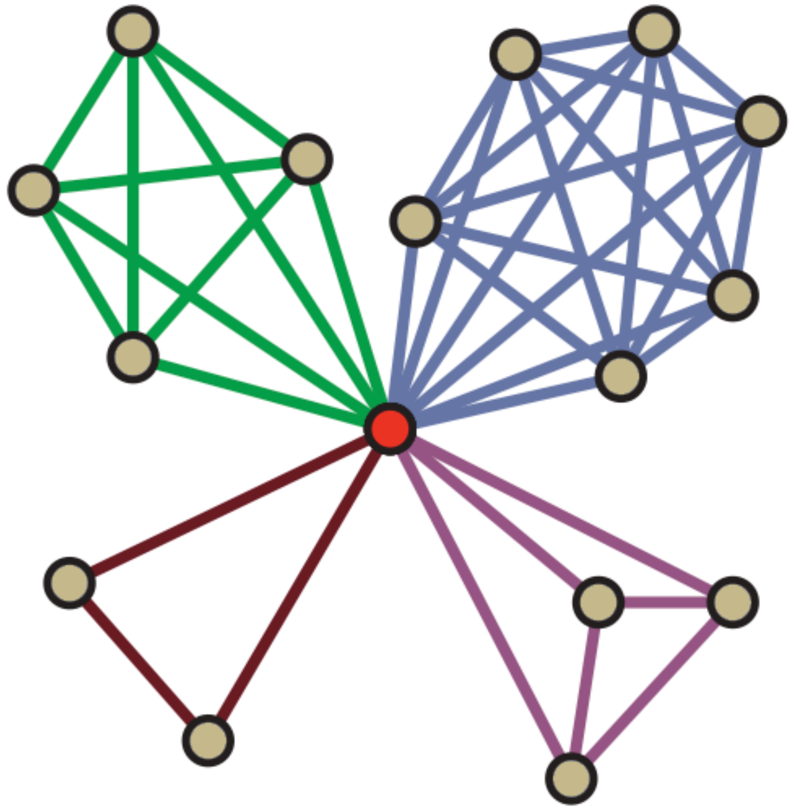
Overlapping communities may exist at different resolutions.

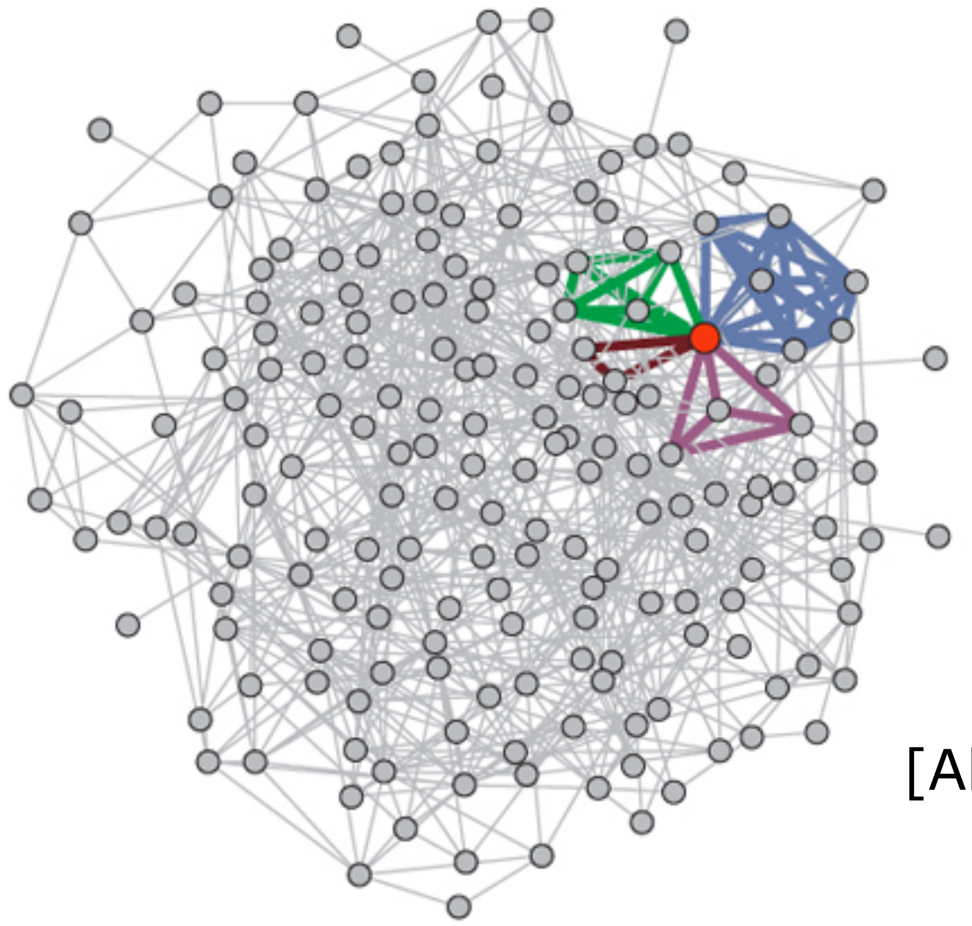


Overlapping v Non-Overlapping

- Distinct "non-overlapping" communities rarely exist at large scales in many empirical networks [Leskovec et al, 2008].

➡ Communities overlap pervasively, making it impossible to partition the networks without splitting communities [Reid et al, 2011].

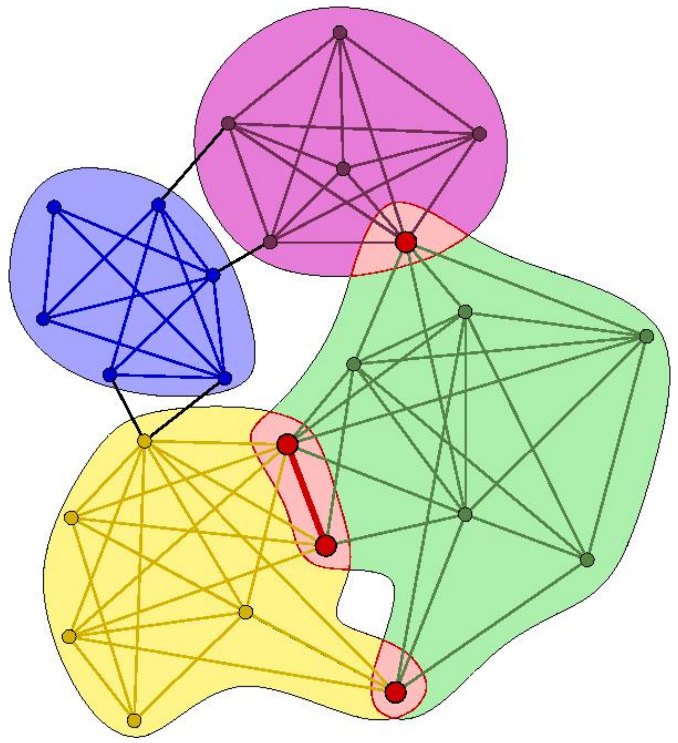

Community overlap at an ego level


Community overlap at a global level

[Ahn et al, 2010]

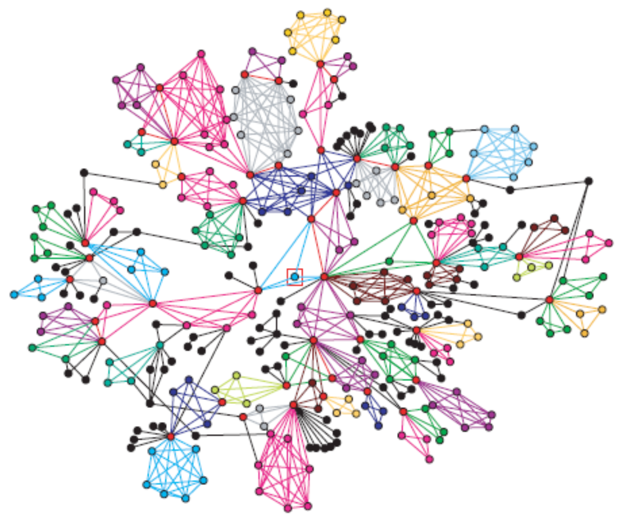
Overlapping Community Finding

- **CFinder**: algorithm based on the clique percolation method [Palla et al, 2005].
- Identify *k*-cliques: a fully connected subgraph *k* nodes.
- Pair of *k*-cliques are "adjacent" if they share *k−1* nodes.
- Form overlapping communities from maximal union of *k*-cliques that can be reached from each other through adjacent *k*-cliques.



Set of overlapping communities
built from 4-cliques.

Co-authorship Network



[Palla et al, 2005]

http://cfinder.org

Overlapping Community Finding

- **Greedy Clique Expansion (GCE)**: identify distinct cliques as seeds, expands the seeds by greedily optimising a local fitness function [Lee et al, 2010].

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

- **MOSES**: scalable approach for identifying highly-overlapping communities [McDaid et al, 2010].
  - Randomly select an edge, greedily expand a community around the edge to optimise an objective function.
  - Delete "poor quality" communities.
  - Fine-tune communities by re-assigning individual nodes.

  https://sites.google.com/site/aaronmcdaid/moses

NetworkX - Overlapping Communities

- No built-in support for overlapping algorithms, but we can use the MOSES tool to analyse graphs represented as edge lists.

```
import networkx
g = networkx.watts_strogatz_graph(60, 8, 0.3)
edgepath = "test_moses.edgelist"
networkx.write_weighted_edgelist(g, edgepath)
```

Build a graph, write it to a temporary edge-list file.

Apply MOSES tool to the edge-list file

```
import subprocess
outpath="test_moses.comms"
proc = subprocess.Popen(["/usr/bin/moses", edgepath, outpath])
proc.wait()
```

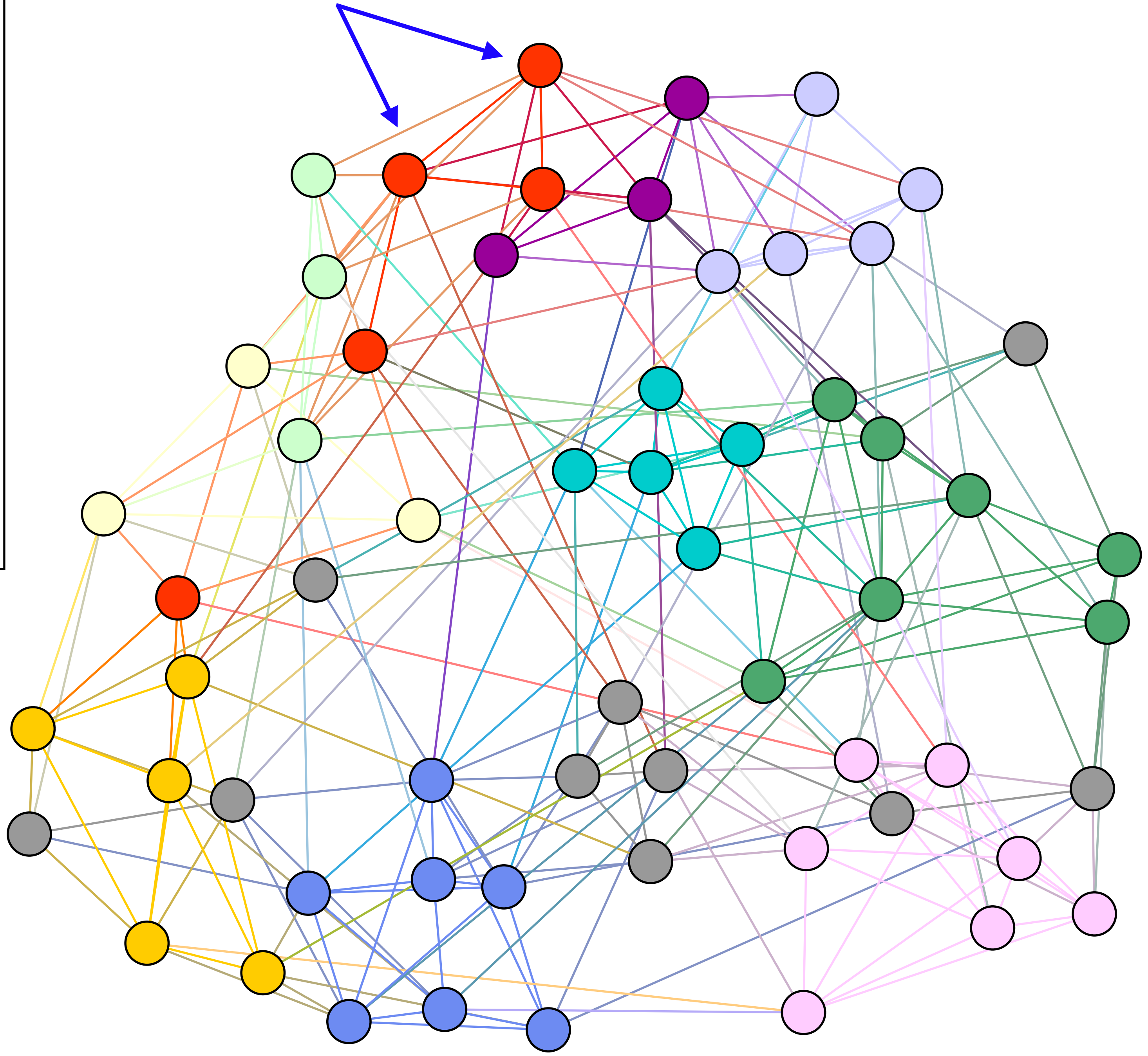
Parse the output of MOSES

```
lines = open(outpath,"r").readlines()
print "Identified %d communities" % len(lines)
for l in lines:
   print set(lines[i].strip().split(" "))
```

NetworkX - Overlapping Communities

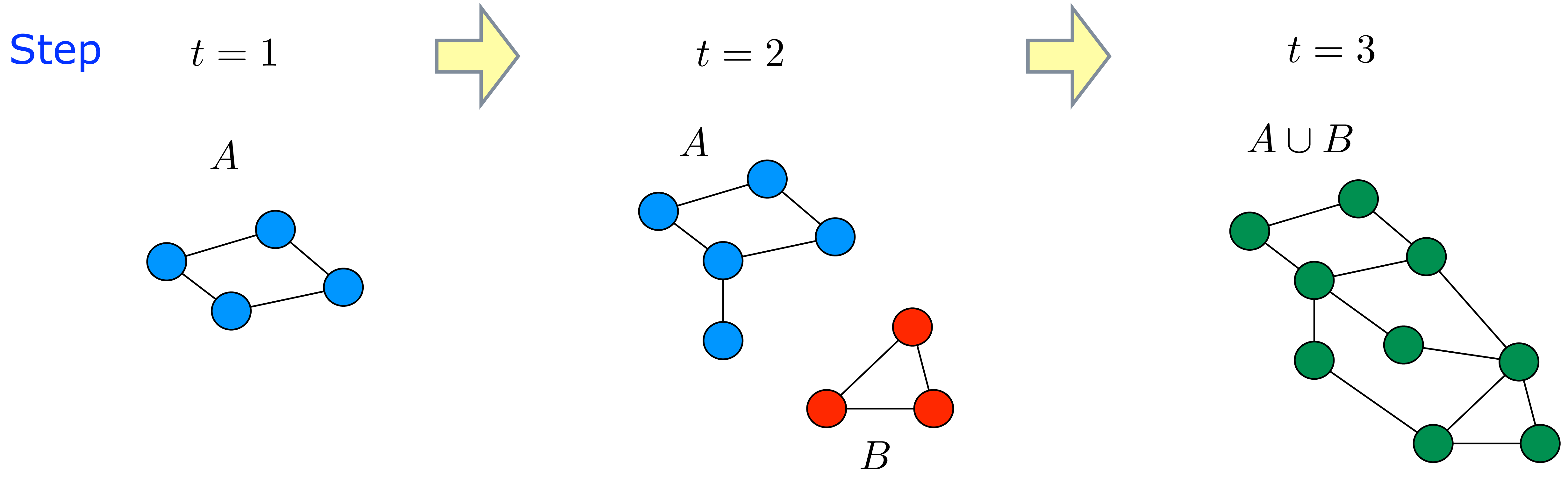
```
Identified 9 communities
Community 0
set(['48', '49', '46', '47', '45', '51', '50'])
Community 1
set(['54', '56', '51', '53', '52'])
Community 2
set(['39', '38', '37', '42', '40', '41'])
Community 3
set(['20', '21', '17', '16', '19', '18', '15'])
Community 4
set(['33', '32', '36', '35', '34'])
Community 5
set(['48', '46', '44', '45', '43', '40'])
Community 6
set(['59', '58', '56', '0', '3', '2'])
Community 7
set(['24', '25', '26', '27', '31', '30', '28'])
Community 8
set(['10', '5', '4', '7', '6', '9', '8'])
```

Nodes assigned to multiple communities



Dynamic Community Finding

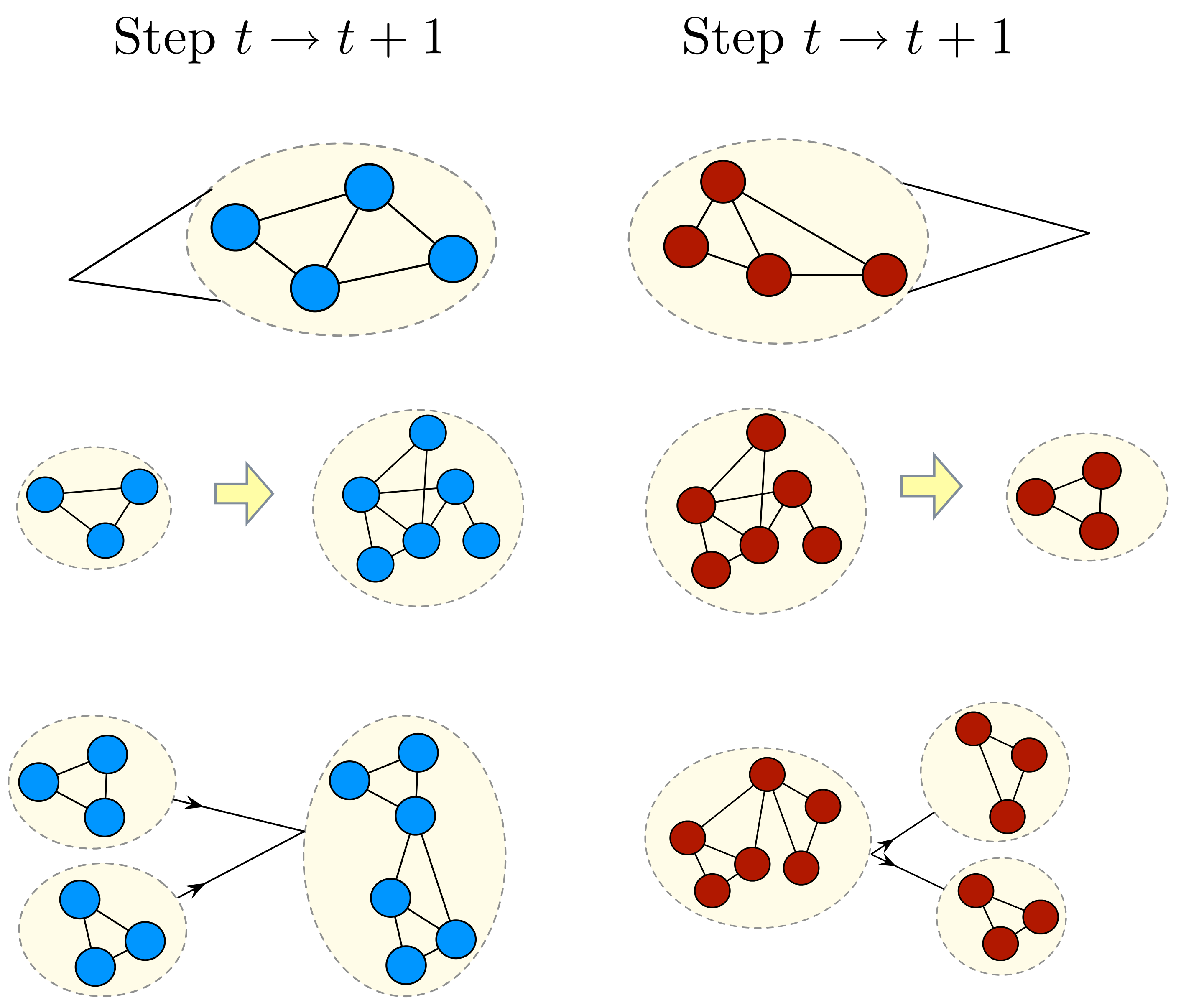
- In many SNA tasks we will want to analyse how communities in the network form and evolve over time.
- Often perform this analysis in an "offline" manner by examining successive snapshots of the network.



Dynamic Community Finding

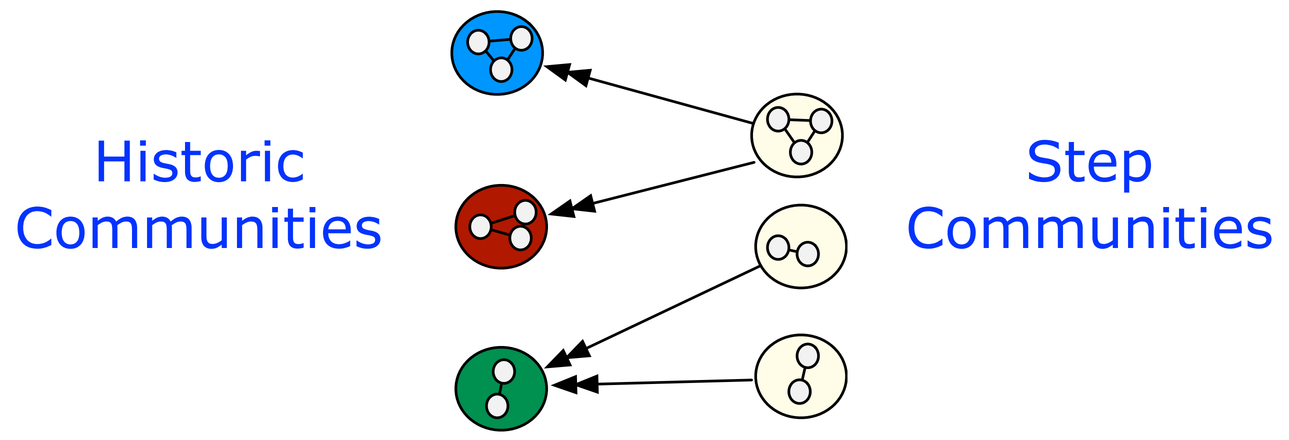
- We can characterise dynamic communities in terms of key life-cycle events [Palla et al, 2007; Berger-Wolf et al, 2007]

  - Birth & Death of communities
  - Expansion & Contraction of communities
  - Merging & Splitting of communities



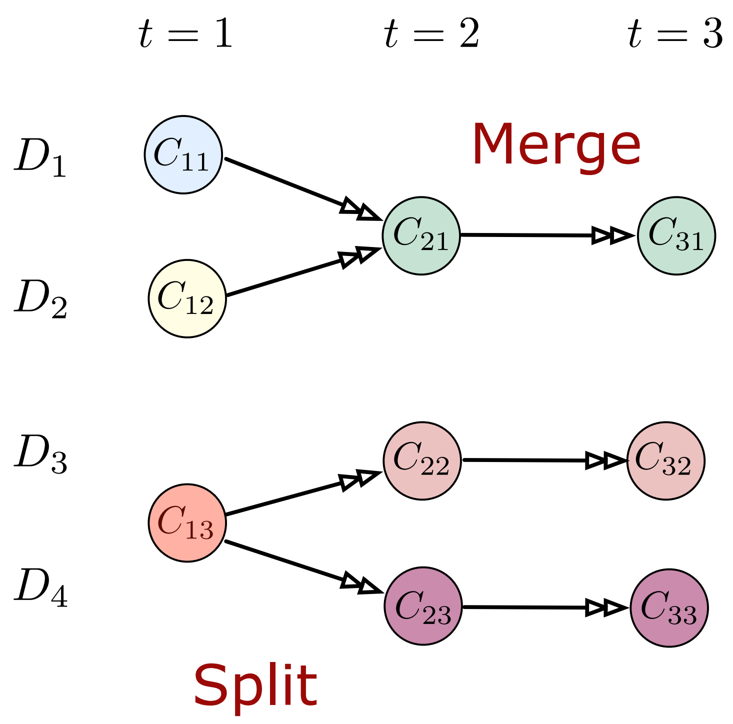
Dynamic Community Finding

- Apply community finding algorithm to each snapshot of the graph.
- Match newly-generated "step communities" with those that have been identified in the past.



Jaccard Similarity Score

$$\frac{|C \cap F_i|}{|C \cup F_i|} > \theta$$



Dynamic community tracking software

http://mlg.ucd.ie/dynamic

[Greene et al, 2010]

Applications

Application - Mapping Blog Networks

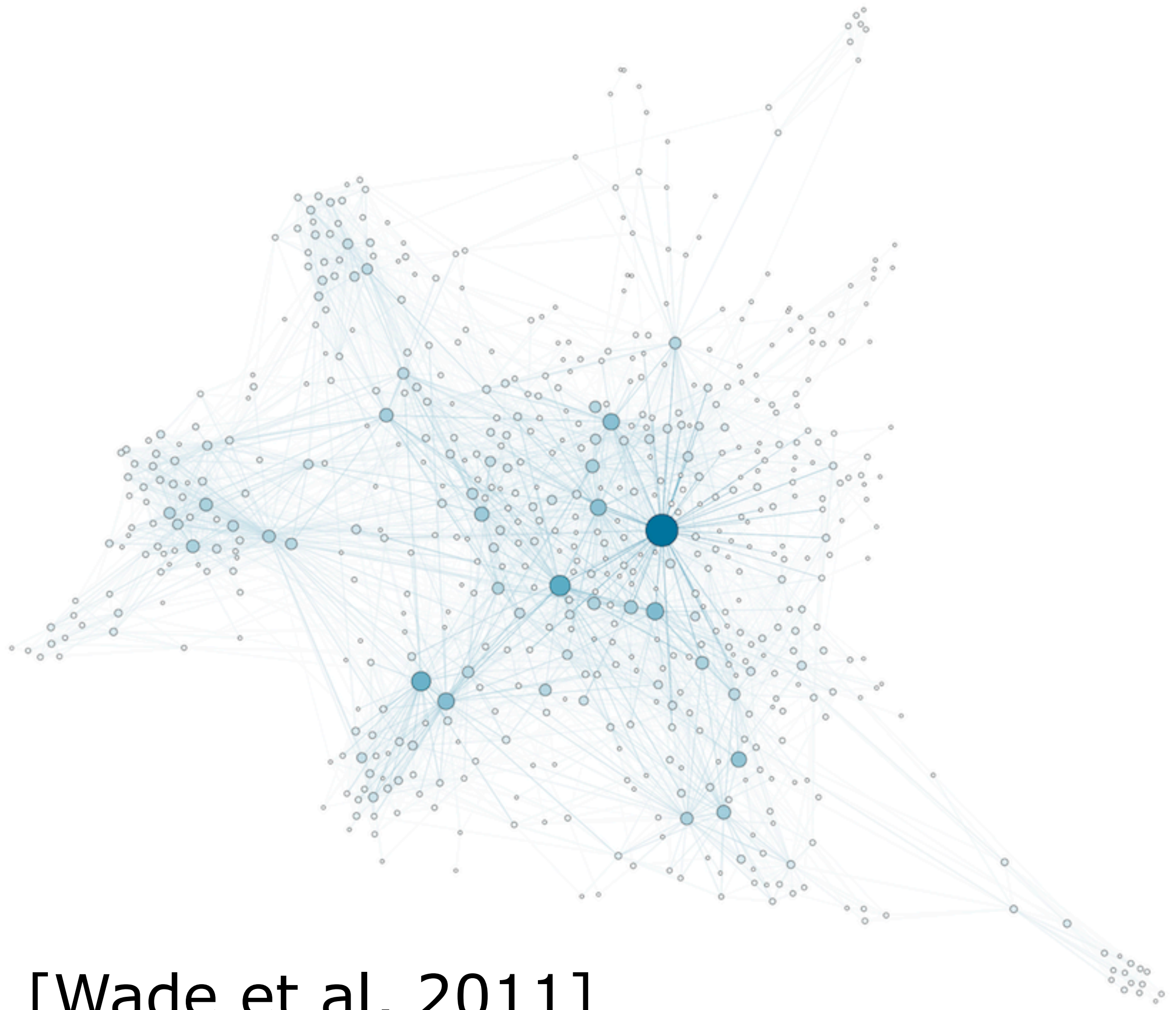
- **Motivation:** Literary analysis of blogs is difficult when corpus contains hundreds of blogs and hundreds of thousands of posts.

➡ Use a data-driven approach to select a topically representative set from 635 blogs in the Irish blogosphere during 1997-2011.

Multi-Relational Networks

1. Blogroll: unweighted graph with edges representing permanent or nearly-permanent links between blogs.
2. Post-link: weighted graph with edges representing non-permanent post content links between blogs.
3. Content profile: text content from all available posts for a given blog.

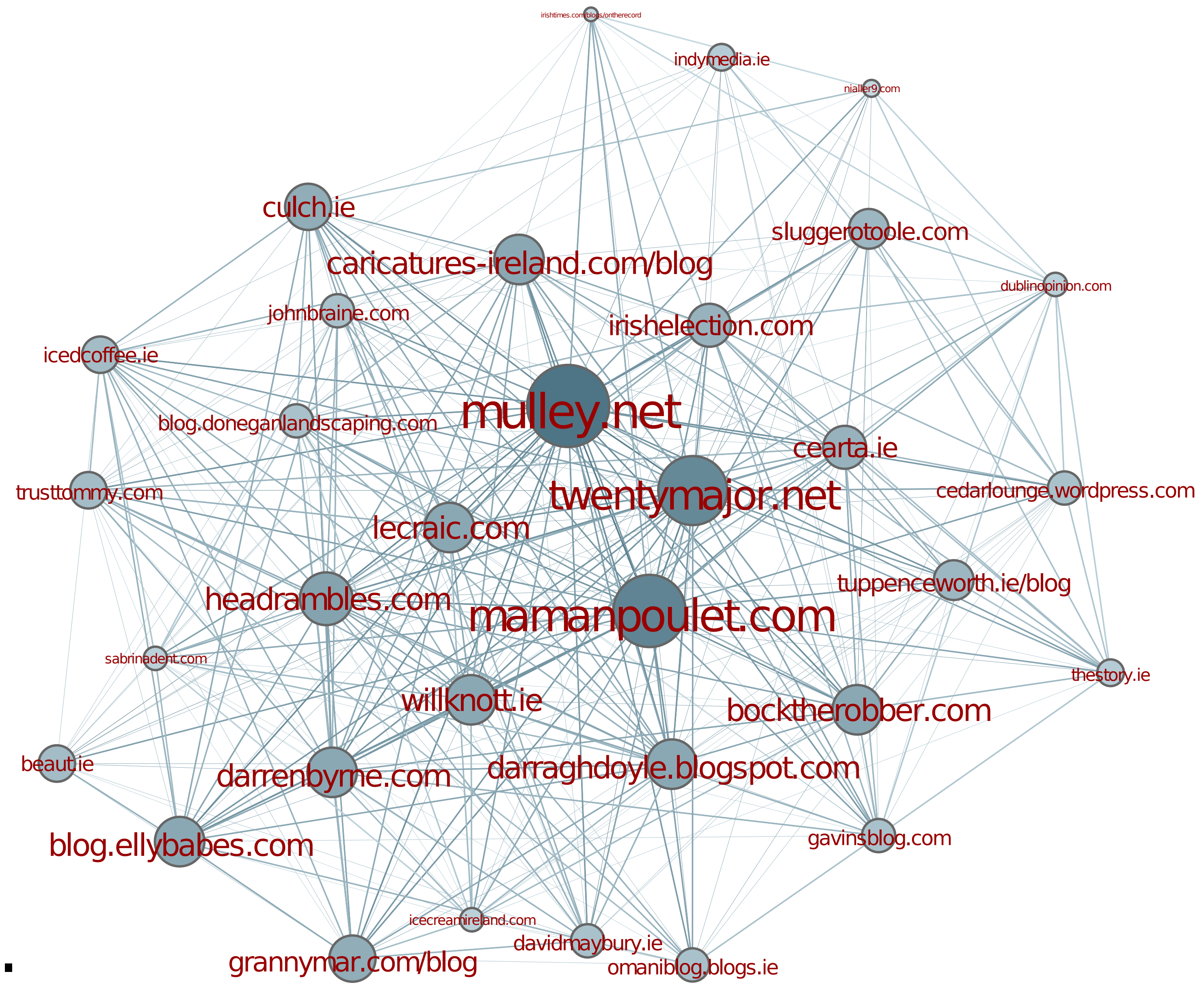
http://mlg.ucd.ie/blogs



[Wade et al, 2011]

Application - Mapping Blog Networks

- Initially applied centrally measures to identify representatives blogs in blogroll and post-link graphs.

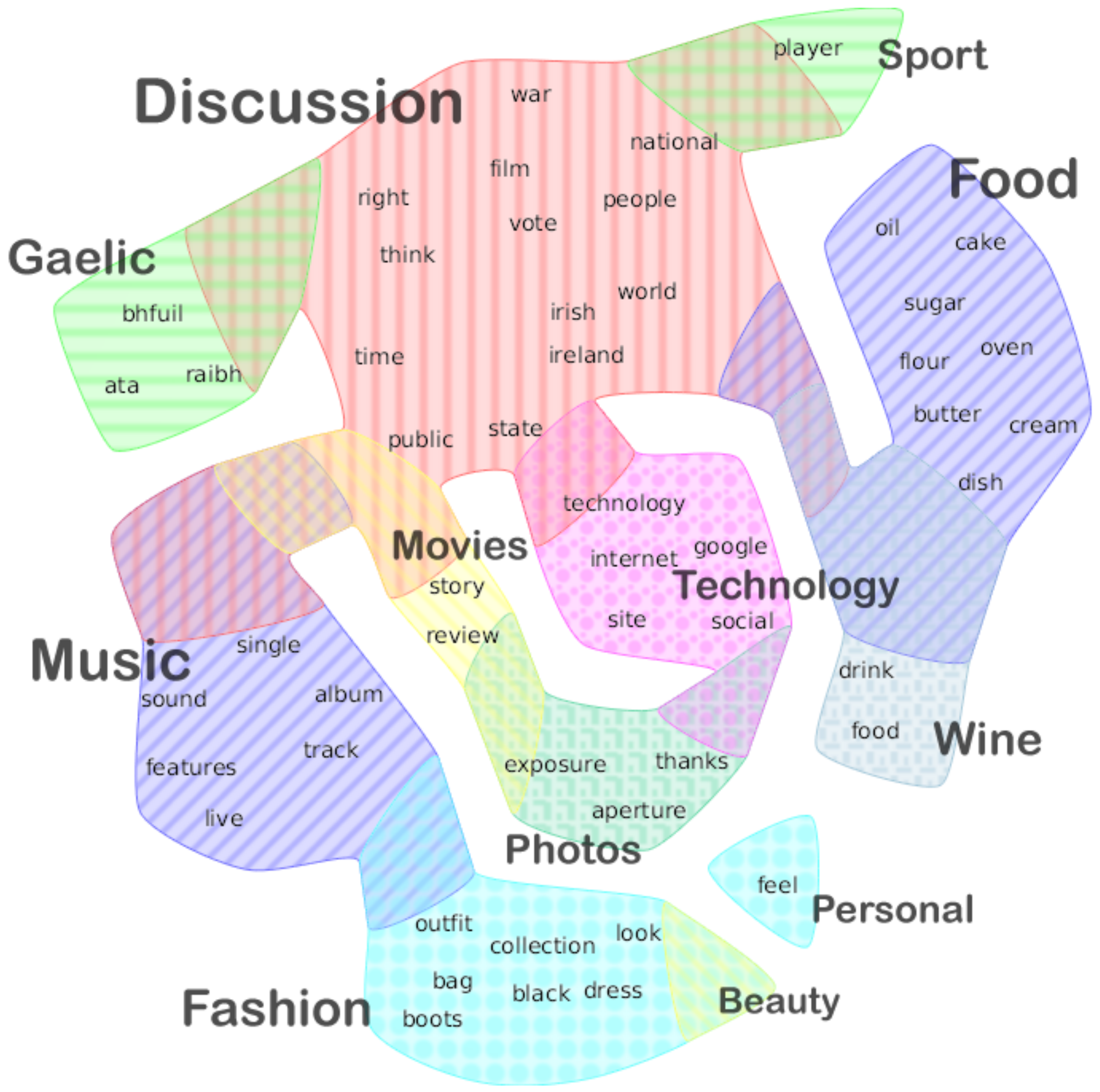


<u>Limitations:</u>

Influential blogs are identified. But they do not necessarily provide good coverage of the wider Irish blogosphere.

Application - Mapping Blog Networks

- Apply text clustering techniques on content view to identify clusters of blogs discussing coherent topics.
- Generated a clustering with 12 distinct communities...

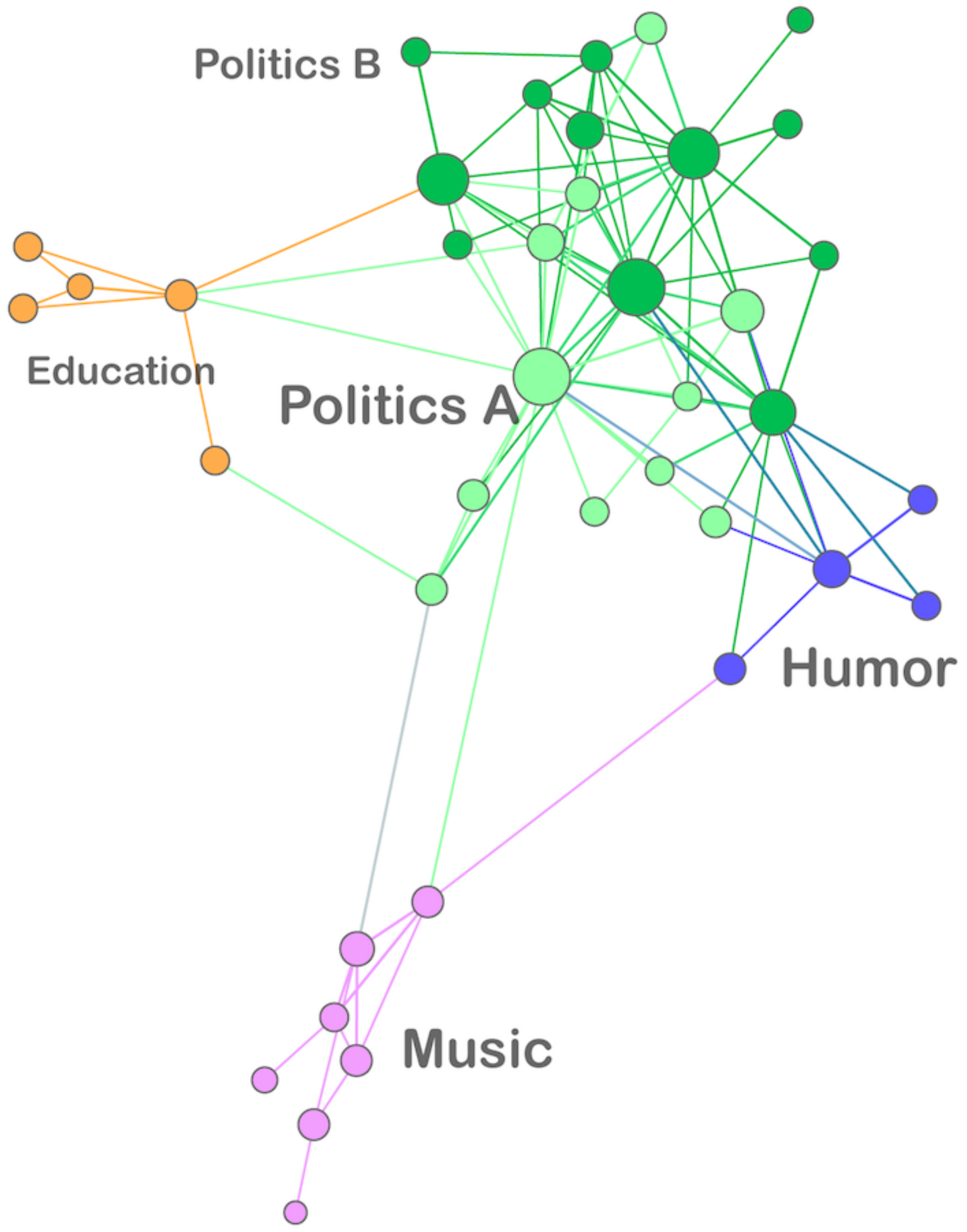


- The "Discussion" community was least coherent in terms of content, and includes blogs pertaining to a number of topics.

Application - Mapping Blog Networks

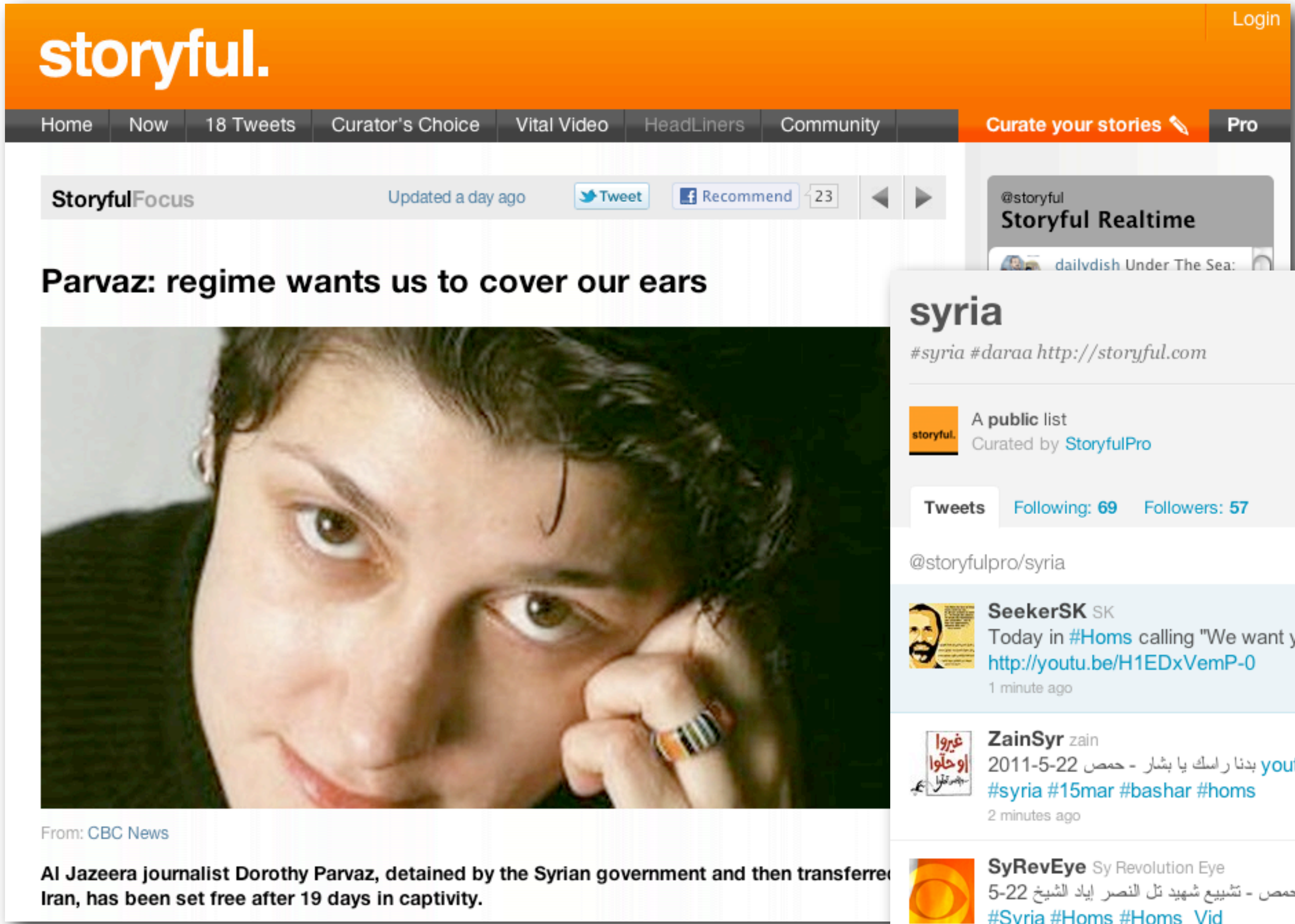
- Applied GCE overlapping community finding algorithm to the blogroll and post-link graphs on the subgraphs induced by the "Discussion" cluster.
- Identified "stable" communities that were present in both the blog-roll and post-link networks.

➡ Identified representative blogs as those with high in-degree in the blogroll subgraphs for each of the resulting communities.



Application - Microblogging

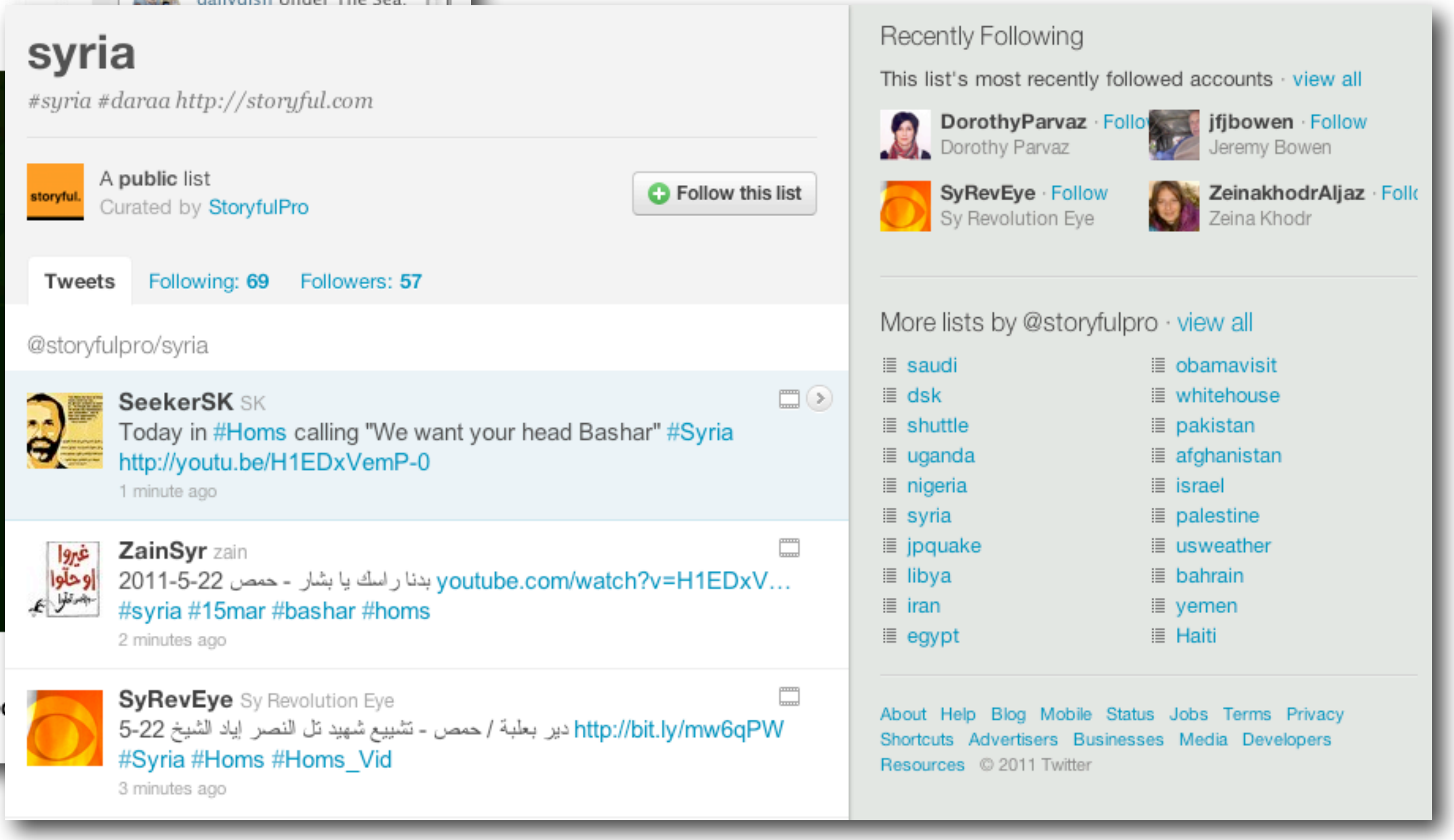
- **Storyful.com:** Journalists struggling to cope with a "tsunami of user-generated content" on social media networks.



Parvaz: regime wants us to cover our ears

From: CBC News

Al Jazeera journalist Dorothy Parvaz, detained by the Syrian government and then transferre
Iran, has been set free after 19 days in captivity.



syria

#syria #daraa http://storyful.com

A public list
Curated by StoryfulPro

@storyfulpro/syria

SeekerSK SK
Today in #Homs calling "We want your head Bashar" #Syria http://youtu.be/H1EDxVemP-0
1 minute ago

ZainSyr zain
2011-5-22 بدنا راسك يا بشار - حمص youtube.com/watch?v=H1EDxV…
#syria #15mar #bashar #homs
2 minutes ago

SyRevEye Sy Revolution Eye
5-22 دير بعلبة / حمص - تشييع شهيد تل النصر اياد الشيخ http://bit.ly/mw6qPW
#Syria #Homs #Homs_Vid
3 minutes ago

Recently Following

This list's most recently followed accounts · view all

DorothyParvaz · Follo Dorothy Parvaz	**jfjbowen** · Follow Jeremy Bowen
SyRevEye · Follow Sy Revolution Eye	**ZeinakhodrAljaz** · Foll Zeina Khodr

More lists by @storyfulpro · view all

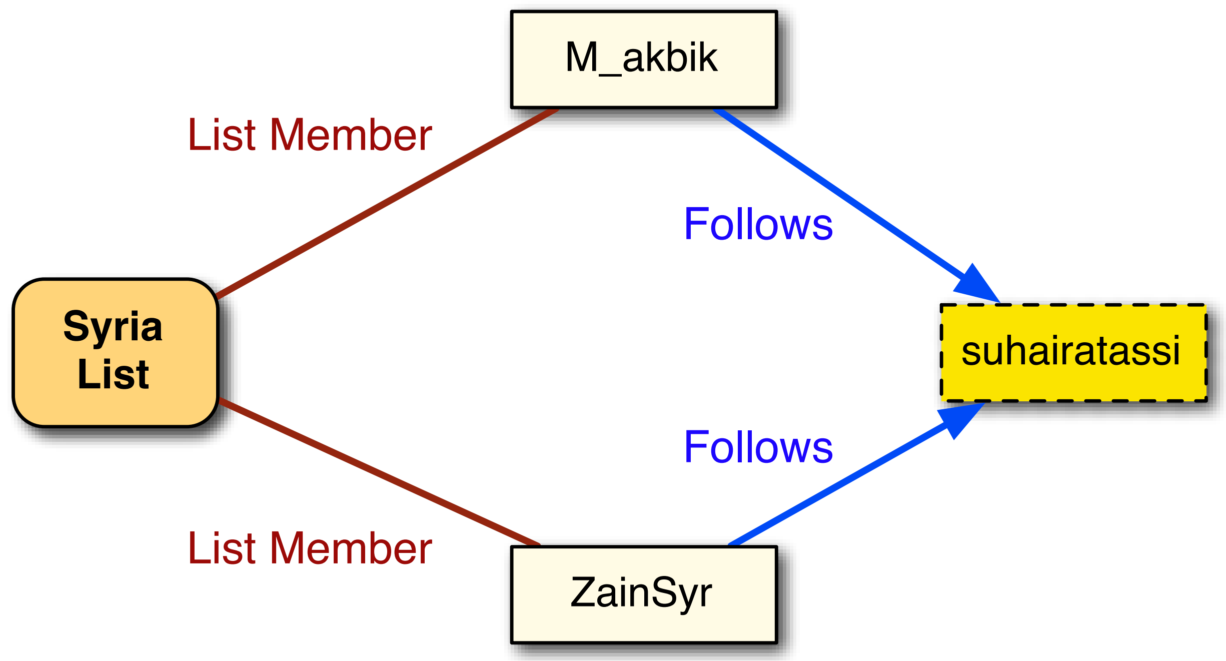
saudi	obamavisit
dsk	whitehouse
shuttle	pakistan
uganda	afghanistan
nigeria	israel
syria	palestine
jpquake	usweather
libya	bahrain
iran	yemen
egypt	Haiti

About Help Blog Mobile Status Jobs Terms Privacy
Shortcuts Advertisers Businesses Media Developers
Resources © 2011 Twitter

Application - Microblogging

- A community of users evolves on Twitter around a breaking news story.

➡ Support the content curation process by augmenting curated lists by recommending authoritative sources from the community that are relevant to the news story.

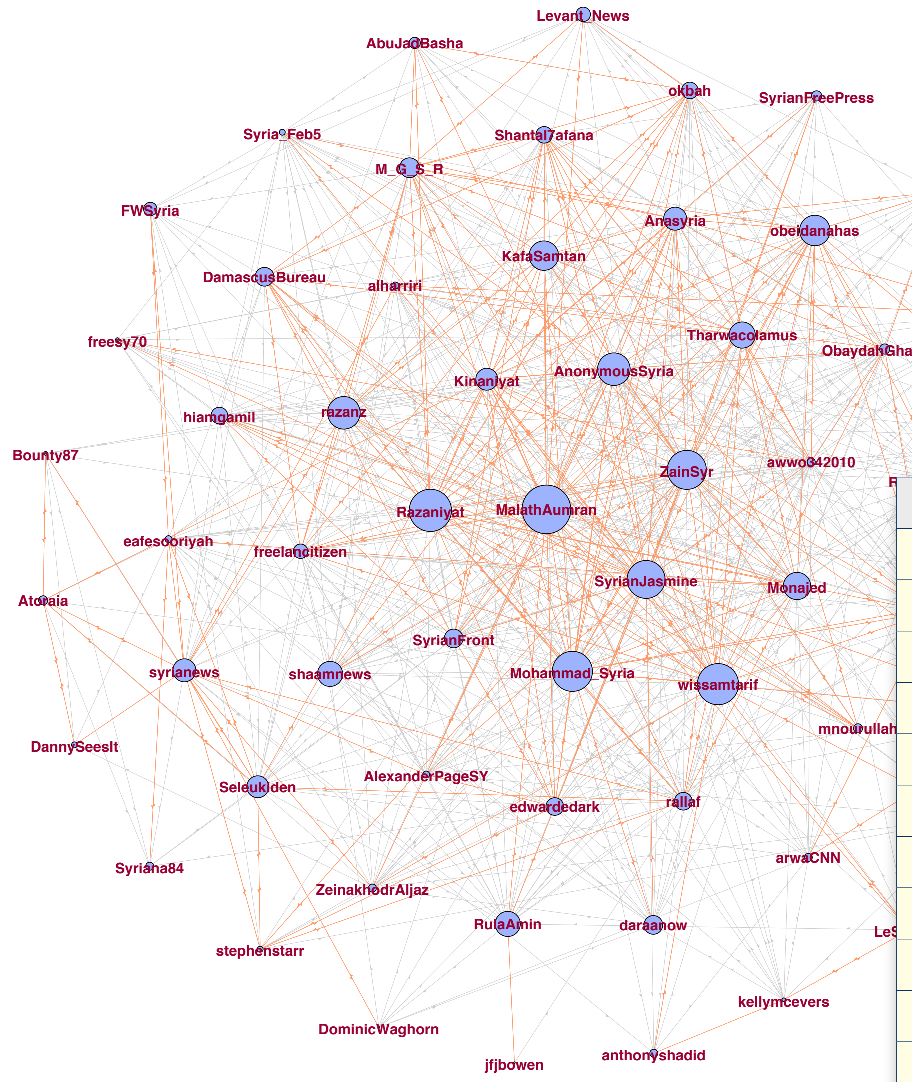


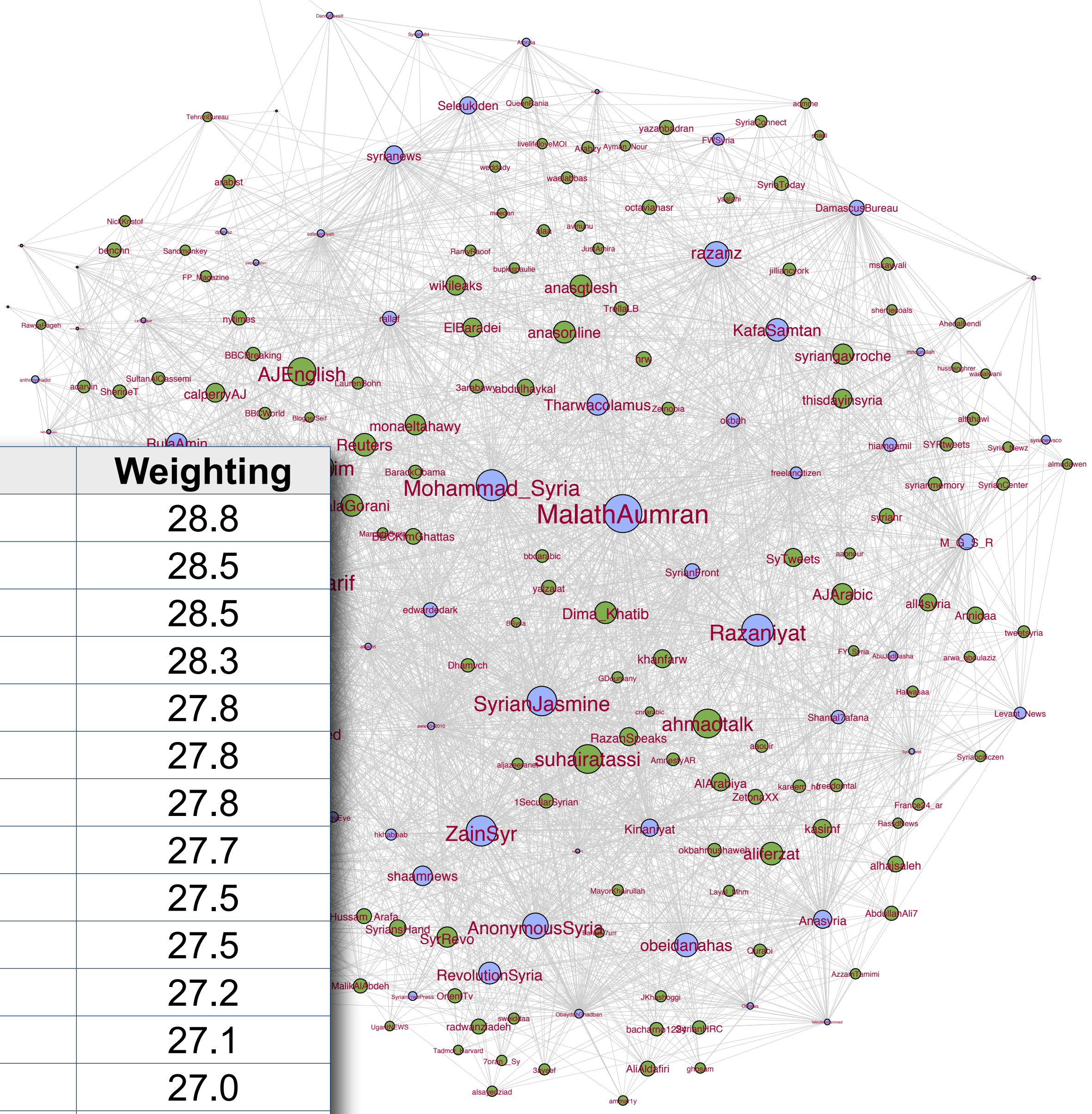
Relations:

- ☐ Follower links
- ☐ Co-listed
- ☐ Shared URLs
- ☐ Retweets
- ☐ Co-location
- ☐ Mentions
- ☐ Content similarity

Application - Microblogging

- Performed analysis to recommend additional Twitter users to augment the "Syria" list curated by Storyful.





Rank	Screen Name	Weighting
1	suhairatassi	28.8
2	aliferzat	28.5
3	syriangavroche	28.5
4	Annidaa	28.3
5	ahmadtalk	27.8
6	Dhamvch	27.8
7	bacharno1224	27.8
8	radwanziadeh	27.7
9	alhajsaleh	27.5
10	MalikAlAbdeh	27.5
11	ZetonaXX	27.2
12	SyrianHRC	27.1
13	anasonline	27.0
14	SyTweets	26.8
15	AbdullahAli7	26.7

Tutorial Resources

- **NetworkX**: Python software for network analysis (v1.5)
  http://networkx.lanl.gov
- Python 2.6.x / 2.7.x
  http://www.python.org
- **Gephi**: Java interactive visualisation platform and toolkit.
  http://gephi.org
- Slides, full resource list, sample networks, sample code snippets online here:
  http://mlg.ucd.ie/summer

References

- J. Moreno. "Who shall survive?: A new approach to the problem of human interrelations". Nervous and Mental Disease Publishing Co, 1934.
- D. Easley and J. Kleinberg. "Networks, crowds, and markets". Cambridge Univ Press, 2010.
- R. Hanneman and M. Riddle. "Introduction to social network methods", 2005.
- M. Newman. "Finding community structure in networks using the eigenvectors of matrices", *Phys. Rev.* E 74, 036104, 2006.
- L. Adamic and N. Glance. "The political blogosphere and the 2004 U.S. election: Divided they blog". In *Proc. 3rd International Workshop on Link Discovery*, 2005.
- S. Fortunato. "Community detection in graphs". *Physics Reports*, 486(3-5):75–174, 2010.
- M. Girvan and M. Newman. "Community structure in social and biological networks". *Proc. Natl. Acad. Sci.*, 99(12):7821, 2002.
- I. Dhillon, Y. Guan, and B. Kulis. "Weighted Graph Cuts without Eigenvectors: A Multilevel Approach". IEEE Transactions on Pattern Analysis and Machine Intelligence, 2007.
- M. Newman and M. Girvan. "Finding and evaluating community structure in networks". *Physical review E*, 69(2):026113, 2004.
- D. J. Watts and S. H. Strogatz. "Collective dynamics of ’small-world’ networks". *Nature*, 393(6684):440–442, 1998.

References

- V. Blondel, J. Guillaume, R. Lambiotte, and E. Lefebvre. "Fast unfolding of communities in large networks". *J. Stat. Mech*, 2008.
- W. W. Zachary. "An information flow model for conflict and fission in small groups", *Journal of Anthropological Research* 33, 452-473, 1977.
- J. Leskovec, K. Lang, A. Dasgupta, and M. Mahoney. "Statistical properties of community structure in large social and information networks". In *Proc. WWW* 2008.
- F. Reid, A. McDaid, and N. Hurley. "Partitioning breaks communities". In *Proc.* ASONAM 2011.
- G. Palla, I. Derenyi, I. Farkas, and T. Vicsek. "Uncovering the overlapping community structure of complex networks in nature and society". *Nature*, 2005.
- C. Lee, F. Reid, A. McDaid, and N. Hurley. "Detecting highly overlapping community structure by greedy clique expansion". In *Workshop on Social Network Mining and Analysis*, 2010.
- A. McDaid and N. Hurley. "Detecting highly overlapping communities with Model-based Overlapping Seed Expansion". In *Proc. International Conference on Advances in Social Networks Analysis and Mining*, 2010.
- Y.-Y. Ahn, J. P. Bagrow, and S. Lehmann. "Link communities reveal multiscale complexity in networks". *Nature*, June 2010.

References

- C. Tantipathananandh, T. Berger-Wolf, and D. Kempe, "A framework for community identification in dynamic social networks". In *Proc. KDD 2007*.
- G. Palla, A. Barabasi, and T. Vicsek, "Quantifying social group evolution ". Nature, vol. 446, no. 7136, 2007.
- D. Greene, D. Doyle, and P. Cunningham. "Tracking the evolution of communities in dynamic social networks". In *Proc. International Conference on Advances in Social Networks Analysis and Mining*, 2010.
- K. Wade, D. Greene, C. Lee, D. Archambault, and P. Cunningham. "Identifying Representative Textual Sources in Blog Networks". In *Proc. 5th International AAAI Conference on Weblogs and Social Media*, 2011.