Overview

What? Detecting clusters in a graph

Why?

- Interesting in their own right
- Communities on the Web; Function modules/groups in PPI or metabolic networks etc..
- Many practical applications
  - Task assignment/partitioning on the web etc...
  - Compression/summarization
Early work in 1927’s

1940’s: George Homans – rearrange the matrix that describe social ties until it is approximately block-diagonal

Graph partitioning – much work in 1970’s on

Seminal work by Girvan and Newman in 2002
  - Many physicists involved in this work since
Examples

- Some nice examples in the "reference" paper
  - Karate Club Social Network – used as a benchmark for a long time
  - Collaboration networks
  - Bottlenose dolphins etc...
  - Protein-protein interaction networks – Cliques indicate functional groups
  - Word association network (next slide)
Fig. 4. Community structure in technological networks. Sample of the web graph consisting of the pages of a web site and their mutual hyperlinks, which were detected with the algorithm of Girvan and Newman (Section 10.2) or by neglecting the directedness of the edges. Reprinted figure with permission from Ref. [27]. © 2003, by the American Physical Society.

Fig. 5. Overlapping communities in a network of word association. The groups, labeled by the colors, were detected with the Clique Percolation Method by Palla et al. (Section 11.1). Reprinted figure with permission from Ref. [28]. © 2005, by the Nature Publishing Group.
Some key challenges

- Scale
- Overlapping communities – hard partitioning isn’t always appropriate
- Reciprocity of edges – undirected vs directed graphs
  - Directed graphs much tougher
- The problem is not well-defined
  - Often the communities are defined algorithmically – whatever the algorithm generated
Defining communities: Local Definitions

- Complete Mutuality: Each community must be clique
  - Too strict
  - Require instead that all nodes in the community are close to each other (say within 2 hops)
  - Has its own problems
    - e.g., are we allowed to go outside the community for the paths?
    - or, a community might be disconnected
- Many definitions that try to balance these
  - n-clique: distance between any pair of nodes is at most n
  - n-clan: same as above, but must use community edges
  - k-plex: each vertex is adjacent to all but k of the vertices in the community
  - k-core: each vertex is adjacent to at least k of the vertices in the community
- All of above are essentially NP-Hard
Defining communities: Local Definitions

- The above definitions don’t care about the edges going outside
  - We need those to be weak interactions, and should incorporate them somehow

- LS-set or Strong Community
  - Subgraph s.t. for each node, internal degree > external degree

- Relaxation
  - Internal degree of the subgraph as a whole > external degree

- Several other definitions, tweaks on the above
Defining communities: Vertex Similarity

- Two vertices are "similar" if they share neighborhoods
- A cluster consists of "similar" vertices
- Many ways to define similarity
  - e.g., directly check the overlap between neighborhoods
  - Based on random walks
    - Commute time: Average number of vertices visited for a random walk to go from $u$ to $v$ and then back to $u$
    - Higher commute time $\rightarrow$ higher dissimilarity
Somewhat trickier

Compare graph being considered to a "random graph"
  ... where every edge is equally likely

By itself does not give an algorithm to find communities

But consider the following:
  Given a graph G
  Say we somehow generate a random graph using this graph as a guideline
  For a group of nodes, compare the # edges within the group to the expected value under the randomization
  A group of nodes is a community if the first number is much larger than the second number

Called "Modularity"
Partition Quality Functions

- Given a partition of a graph, assign a "score" to it so we can talk about the best partition
- Performance and coverage: some function of # of good edges (intra-cluster) and bad edges (inter-cluster)
- Modularity (Girvan and Newman)

\[
Q = \frac{1}{2m} \sum_{ij} (A_{ij} - P_{ij}) \delta(C_i, C_j)
\]

- \(m\) = number of edges
- \(C_i\) = community of vertex \(i\)
- \(\delta(C_i, C_j) = 1\) if the communities are identical
- \(A_{ij} = 1\) if there is an edge between \(i\) and \(j\) in the original graph
- \(P_{ij}\) = "probability" that there is an edge between \(i\) and \(j\) in a "random graph"
Given a partition of a graph, assign a "score" to it so we can talk about the best partition.

Performance and coverage: some function of # of good edges (intra-cluster) and bad edges (inter-cluster).

Modularity (Girvan and Newman)

Formula:

\[ Q = \frac{1}{2m} \sum_{ij} (A_{ij} - P_{ij}) \delta(C_i, C_j) \]

- \( m \) = number of edges; \( C_i \) = community of vertex \( i \)
- \( \delta(C_i, C_j) = 1 \) if the communities are identical
- \( A_{ij} = 1 \) if there is an edge between \( i \) and \( j \) in the original graph
- \( P_{ij} = "probability" \) that there is an edge between \( i \) and \( j \) in a "random graph"
Modularity

Modularity (Girvan and Newman)

Formula:

\[ Q = \frac{1}{2m} \sum_{ij} (A_{ij} - P_{ij}) \delta(C_i, C_j) \]

- \( m \) = number of edges; \( C_i \) = community of vertex \( i \)
- \( \delta(C_i, C_j) = 1 \) if the communities are identical
- \( A_{ij} = 1 \) if there is an edge between \( i \) and \( j \) in the original graph
- \( P_{ij} = \) "probability" that there is an edge between \( i \) and \( j \) in a "random graph"

A partition is rewarded if \( i \) and \( j \) are in a cluster, there is an edge between \( i \) and \( j \) and in the random graph, and probability of that edge is low

So, the edge affected the partitioning and that's good

Everything depends on the "random graph" model chosen to compare (called null model)
Modularity: "Null" model

- (1) Same number of edges as the original graph, but the edges are assigned to the pairs of vertices at random
  - In that case, $P_{ij} = 2m/n(n-1)$ is a constant
  - Not very good: The degree distribution of the random graph is too different

- (2) Prefer to keep the same degree distribution as the original graph
  - Turns out this can be defined w/o much hassle
  - Also, only required to be true in expectation
  - Gives: $P_{ij} = k_i k_j / 2m$, where $k_i =$ degree of $i$
  - Can be rewritten in terms of inter-cluster edges in the partition
  - Modularity can be negative: consider a partition where every node is in a separate partition
Graph Partitioning

- Rich literature in computer science
  - Originally in circuit partitioning
- Usually look for "bisections" or "balanced partitions"
- Most problems NP-Hard, so heuristics often used
  - Very nice packages available to partition large graphs or hypergraphs
- Spectral clustering
  - A set of techniques based on eigenvalues of graph’s Laplacian matrix
Community Detection: Divisive Algorithms

- Identify the edges that connect different communities, remove, recurse till disconnection
- How to identify inter-community edges?
  - Girvan and Newman
    - Compute the centralities of all edges
    - Remove the edge with highest centrality
    - Recompute centralities; Recurse
- 3 different ways to define edge centrality (used by GN)
  - *Edge betweenness*: # of shortest paths in the graph that go through that edge
  - Intuition: Edges between communities likely have a lot of shortest paths going through them

Fig. 10. The most prominent criterion for disconnection is edge betweenness in a graph. © Springer
Other centrality measures

random-walk betweenness: what is the probability that a random walker will cross an edge when going from $u$ to $v$

- Average over all pairs of nodes to get overall betweenness for the edge

current-flow betweenness: imagine each edge is a resistor

- for a pair $(u, v)$, attach a voltage difference
- compute the current through each edge (Kirchoff’s equations)
- for each edge, average over all pairs of nodes

Turns out edge betweenness works the best, and is fastest

In a later paper, same authors introduced modularity to define goodness of a partition
Community Detection: Divisive Algorithms

- Much later work that followed the basic GN approach
  - Faster algorithms, through use of better data structures, approximations
- Other methods
  - Edge clustering coefficient: # of triangles that pass through an edge
    - Remove the edge with the lowest clustering coefficient; Recurse
  - Several other measures of centrality have been defined
Recall: Modularity = how good is a partition
Why not try to optimize for that directly?
Note: NG introduced and used modularity, but did not directly optimize for it

(1) Greedy algorithm (Newman)
- Start with each vertex in a separate partition
- Merge two clusters together that give the highest increase in modularity
- Many improvements since, but may not be very good still

(2) Simulated Annealing
- A greedy local optimization heuristic
- Appeared to work very well in the experiments

Several other techniques, based on spectral optimization, Mathematical programming etc.
Community Detection: Modularity-based Algorithms

- Many other definitions of modularity
- Limits of Modularity as a Metric
  - A large value of modularity does not necessarily mean there is community structure
  - More important, fundamentally modularity-based optimization has trouble identifying "small" communities relative to the graph size
    - Called "resolution limit"
Community Detection: Other Algorithms

- Spectral Algorithms
  - Many algorithms proposed based on finding eigenvalues of different matrices corresponding to the graphs
    - e.g. Laplacian matrix
  - An example technique
    - Use eigenvector components to project the vertices into a m-dimensional space
    - Do Euclidean clustering in that space

- Dynamic Processes
  - Algorithms based on: spin-spin interactions, random walks, synchronization
  - One based on random walks appeared to best in the experiments on LFR benchmark
    - Intuition: if there are communities, a random walker will spend a lot of time inside the community
  - Many variations on these techniques to find communities
Information theory/Compression-based approach

How best to communicate a network structure?

In other words, find the best summarization

Similar to the MDL-based approach we saw last week

In Fig. 2, we apply our cluster-based compression method to the dolphin social network reported by Lusseau et al. (14). Our method selects a division that differs by only one node from the division along which the actual dolphin groups were observed to split. Because it is computationally infeasible to check all possible partitions of even modestly sized networks, we use simulated annealing with the heat-bath algorithm to search for the partition that maximizes the mutual information between the description and the original network. We have confirmed the results for the networks in the figures with exhaustive searches in the vicinity of the Monte Carlo solutions.

We compare our results with the partition obtained by using the modularity approach introduced by Newman and Girvan in ref. 15; that technique has been widely adopted because of its appealing simplicity, its performance in benchmark tests (10), and the availability of powerful numerical techniques for dealing with large networks (16–18). Given a partitioning into \( m \) modules, the modularity \( Q \) is the sum of the contributions from each module:

\[
Q = \sum_{i} \left( \frac{n_i}{d_i} - \frac{l_{ii}}{l} \right)
\]

where \( n_i \) is the number of nodes in module \( i \), \( d_i \) is the total degree in module \( i \), and \( l \) is the total number of links.
Information theory/Compression-based approach

More sophisticated, but also works for directed graphs

- The choice between this approach, their prior approach, and a modularity-based approach really application dependent
- Depends on what you want to do with the communities and what the links represent

Basic idea

- Imagine a random walk on the graph
- Say I want to communicate this random walk to someone else
- I would like to use the minimum number of bits
Fig. 1. Detecting communities by compressing the description of information flows on networks. (A) We want to describe the trajectory of a random walk on the network such that important structures have unique names. The orange line shows one sample trajectory. (B) A basic approach is to give a unique name to every node in the network. The Huffman code illustrated here is an efficient way to do so. The 314 bits shown under the network describe the sample trajectory in A, starting with 1111100 for the first node on the walk in the upper left corner, 1100 for the second node, etc., and ending with 00011 for the last node on the walk in the lower right corner. (C) A two-level description of the random walk, in which major clusters receive unique names, but the names of nodes within clusters are reused, yields on average a 32% shorter description for this network. The codes naming the modules and the codes used to indicate an exit from each module are shown to the left and the right of the arrows under the network, respectively. Using this code, we can describe the walk in A by the 243 bits shown under the network in C. The first three bits 111 indicate that the walk begins in the red module, the code 0000 specifies the first node on the walk, etc. (D) Reporting only the module names, and not the locations within the modules, provides an efficient coarse graining of the network.
This doesn’t give any communities though

However, observe we can use even shorter codewords if we find communities and reuse codewords within the community
Fig. 1. Detecting communities by compressing the description of information flows on networks. (A) We want to describe the trajectory of a random walk on the network such that important structures have unique names. The orange line shows one sample trajectory. (B) A basic approach is to give a unique name to every node in the network. The Huffman code illustrated here is an efficient way to do so. The 314 bits shown under the network describe the sample trajectory in A, starting with 1111100 for the first node on the walk in the upper left corner, 1100 for the second node, etc., and ending with 00011 for the last node on the walk in the lower right corner. (C) A two-level description of the random walk, in which major clusters receive unique names, but the names of nodes within clusters are reused, yields on average a 32% shorter description for this network. The codes naming the modules and the codes used to indicate an exit from each module are shown to the left and the right of the arrows under the network, respectively. Using this code, we can describe the walk in A by the 243 bits shown under the network in C. The first three bits 111 indicate that the walk begins in the red module, the code 0000 specifies the first node on the walk, etc. (D) Reporting only the module names, and not the locations within the modules, provides an efficient coarse graining of the network.

Rosvall and Bergstrom PNAS / January 29, 2008 / vol. 105 / no. 4 / 1119
APPLIED MATHEMATICS
Difficulties in testing the algorithms

*Planted l-partition model:* Artificially create a graph with known communities

This paper extends the model to do power law distributions and heterogenous sizes

- Needs careful analysis

Aside: if parameters too tight, specific networks generated may not exhibit community structures

Testing communities: Can do using mutual information-like metric

- Why not use modularity?

Infomap (Rosvall, Bergstrom 2) seemed to do the best
Some interesting points

- Networks of communities (after collapsing) have different degree distributions than original graphs (28)
- Work on coarse-graining that keep the properties of dynamic processes like random walks (41-42)
- Kleinberg’s clustering impossibility theorem doesn’t apply because no distance function