CS3230R

Community Detection in graphs

A review paper by

Santo Fortunato

Davin Choo

Outline

- **Introduction and Definitions**
- Approaches
  - Traditional methods
  - Divisive algorithms

Introduction

- Random Graph (Erdos and Renyi)
  - Every pair of vertices have equal probability of having an edge between them
  - “Disordered graph”
- Graphs in real life
  - Clusters
  - “Follow power law”
    (Many low deg vertices, few high deg vertices)

Community Detection

Uses

- Analysing social media: Twitter, Facebook, G+ etc.
- 911 / Terrorism
- Clustering web clients geographically to improve performance
- Identify correlated structures (e.g. PPI networks)
- Classifying people
  (e.g. Group leaders, “Influencers”, “Mediators”, etc.)

Things to take note of

- Requirement: Graph is sparse
- Undirected vs Directed
- Overlapping/Cover vs. Partitioning
- Multipartite graphs
- Unweighted vs. Weighted
- Many clustering algorithms/problems are NP-hard (i.e. no known polynomial time solutions)
    - Approximation algorithms

Defining "community"

- No fixed definition / Vague
- Each algorithm usually optimize over a particular function/ property which they deem important
- Examples
  - Intra-cluster density [$\delta_{int}(\mathcal{C})$ = (# internal edges of $\mathcal{C}$)/(total possible internal edges)]
  - Inter-cluster density [$\delta_{ext}(\mathcal{C})$ = (# inter-cluster edges of $\mathcal{C}$)/(total possible inter-cluster edges)]
  - Average link density [(# edges in graph)/(total possible edges)]
  - Connectedness [$\exists$path between nodes, using only paths in $\mathcal{C}$]

Classes of "community" definition

- Local [Focus on subgraphs]
  - n-clique, n-clan, n-club, k-plex, k-core, etc
- Global [Evaluate entire graph]
  - Null model [Uses idea that "random graph (Erdos/Renyi) has no structure"]
  - Quality functions [Covered later]
- Vertex similarity [Idea: Group similar vertices]
  - Put graph into a metric space / "Walking"

Quality Functions

- Q : Partition $\longmapsto$ Value
- Used to rank different partitioning of graphs
- Additivity property: $Q(\mathcal{P}) = \sum_{\mathcal{C}\in\mathcal{P}} q(\mathcal{C})$, for some function q
- Performance
  [((Edges within partitions) + (Missing edges across partitions))/(Total possible edges)]
- Coverage [(Intra-community edges)/(Total number of edges)]
  i.e. If clusters are disjoint, then coverage = 1

Modularity

- Frequently used Quality Function
- Proposed by Newman and Girvan
- Idea: Random graph not expected to have cluster structure, so possible existence of clusters is revealed by comparison between actual density of edges in a subgraph and a random subgraph
- Many choices for modularity formula and null models

Outline

- Introduction and Definitions
- Approaches
  - **Traditional methods**
  - Divisive algorithms

Traditional Methods

- Graph partitioning
- Hierarchical clustering
- Partitional clustering
- Spectral clustering

Graph partitioning

- Idea:
  - Fix #groups **g**
  - Fix size **s**
  - Find partition such that cut edges (inter-partition edges) are minimised

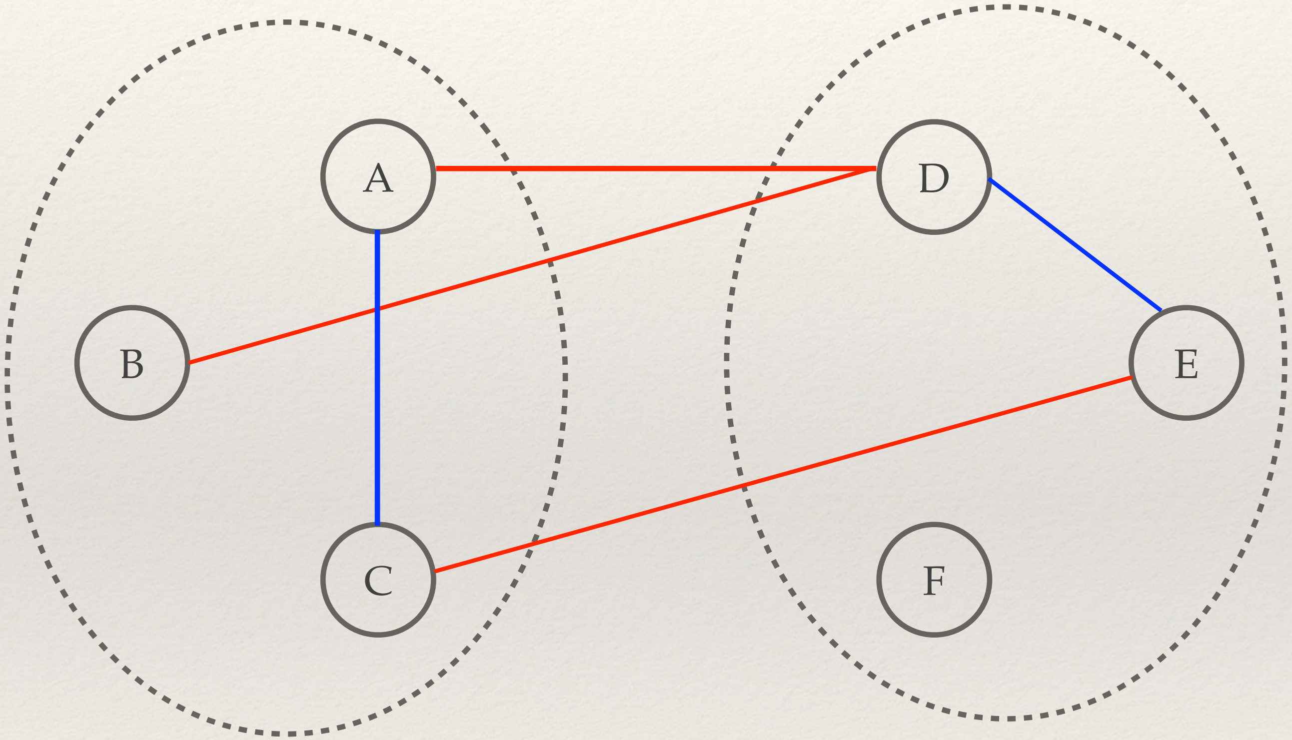
Why?
(Trivial solution)

Kernighan-Lin algorithm

- Input: Graph
- Output: 2 Partitions/Modules
- Optimize over **Q** = "# edges inside partitions - # edges across partitions"
- Randomly initialize 2 partitions with same number of vertices
- Swap vertices between partitions that gives maximal increase on Q repeatedly
- Find out more on Wikipedia: http://en.wikipedia.org/wiki/Kernighan–Lin_algorithm

KL algorithm (Example)

$Q = 2 - 3 = -1$



<u>What happens if we shift D to left partition?</u>

-1 for blue edge introduced across partitions

+2 for red edges removed across partitions

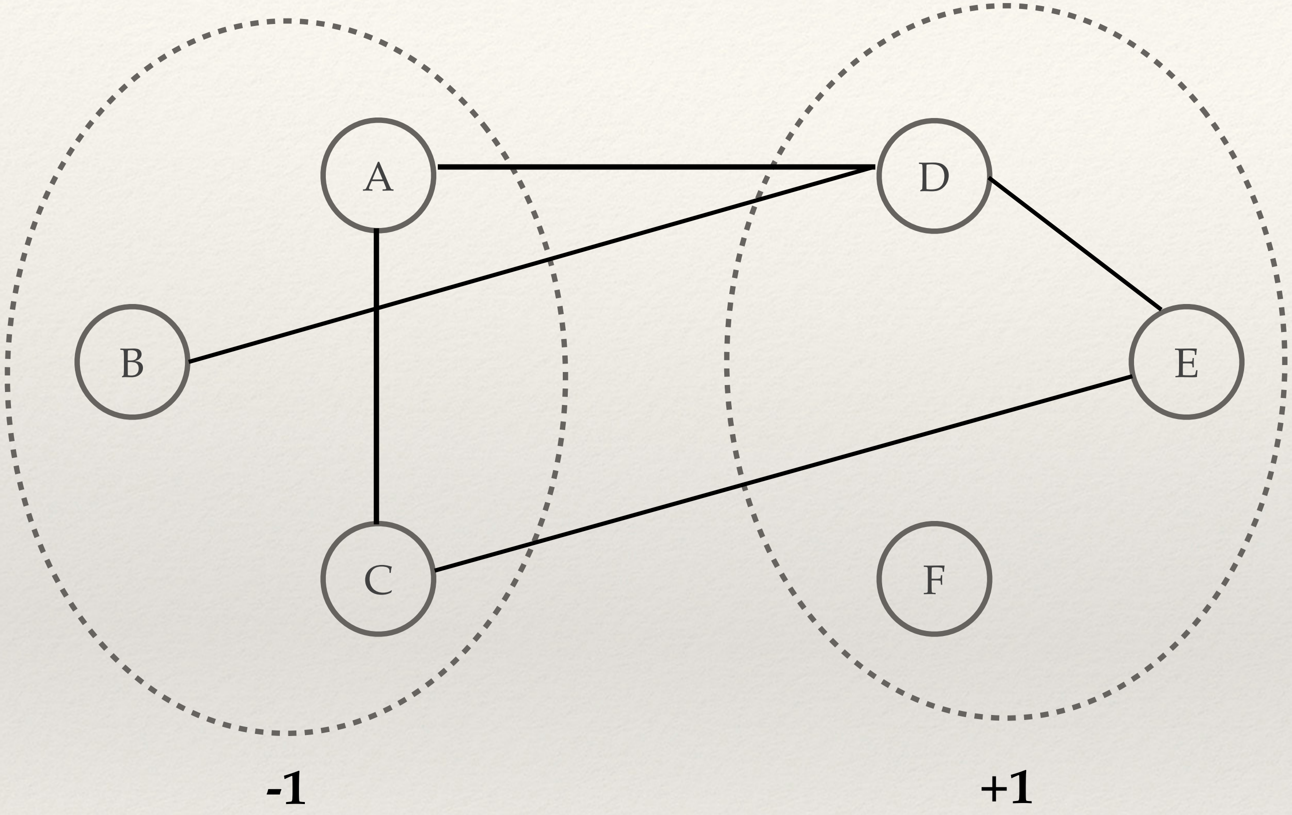
Kernighan-Lin algorithm

- Weakness: Performance dependent on initialisation
- Extensions
    - Weighted graph
    - Directed graph

Spectral Bisection

- Index vector **s**
  - Mark vertices with “+1” or “-1” to indicate partition
- Compute Laplacian matrix **L**
- Cut size $R = \frac{1}{4} s^T L s$ (minimize this)
  - Find eigenvector of **L**
- Find out more: http://www.cs.ucdavis.edu/~bai/ECS231/Graphpartition.pdf (Pg 9-16)

Spectral Bisection (Example)



s

A	-1
B	-1
C	-1
D	1
E	1
F	1

L

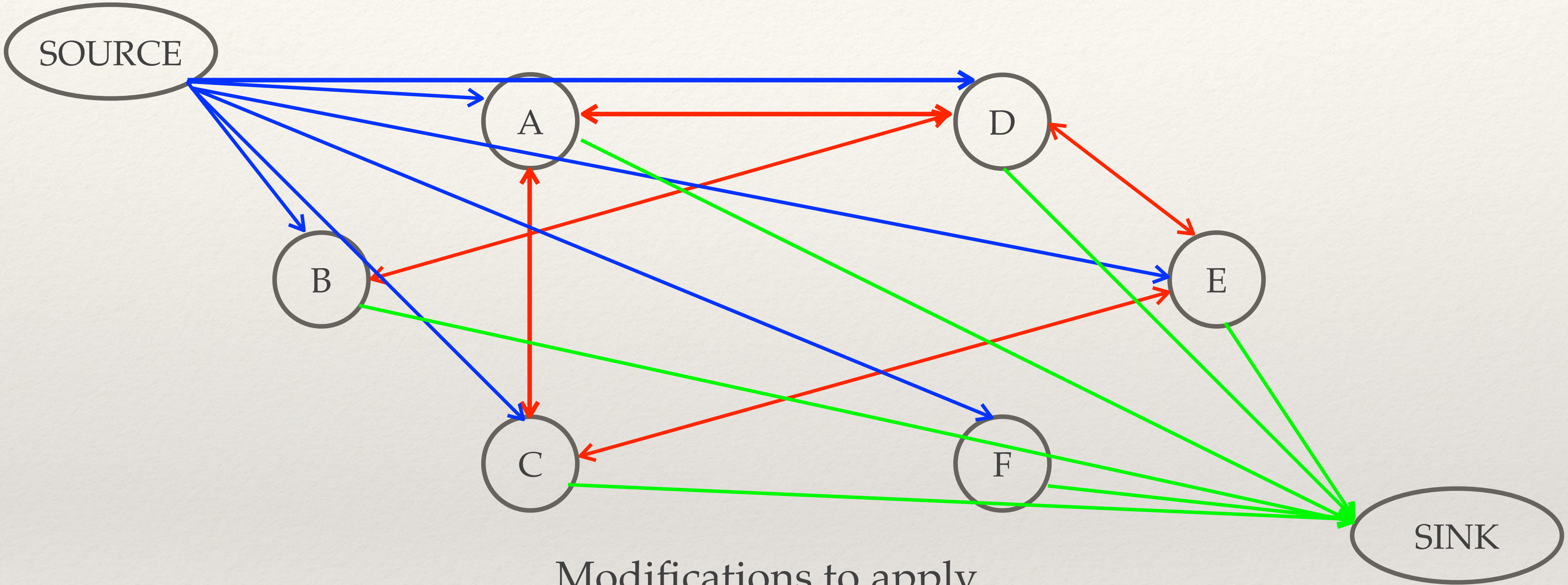
	A	B	C	D	E	F
A	2	0	-1	-1	0	0
B	0	1	0	-1	0	0
C	-1	0	2	0	-1	0
D	-1	-1	0	3	-1	0
E	0	0	-1	-1	2	0
F	0	0	0	0	0	0

$$\ell_{i,j} := \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$

Max-flow Min-cut

- By Ford and Fulkerson
- Input: Flow network (1 source, 1 sink, directed, weighted)
- Output: Set of edges **S** to cut to disjoint source from sink
- Condition: Sum of edge weights of **S** is minimized
- Find out more on Wikipedia: http://en.wikipedia.org/wiki/Max-flow_min-cut_theorem

Max-flow Min-cut (Example)



<u>Modifications to apply</u>

1) Add artificial source
2) Add artificial sink
3) Make undirected edges directed
   (Add directed edge in both directions)
4) Make unweighted graph weighted
   (All edges weight 1)

Other Measures

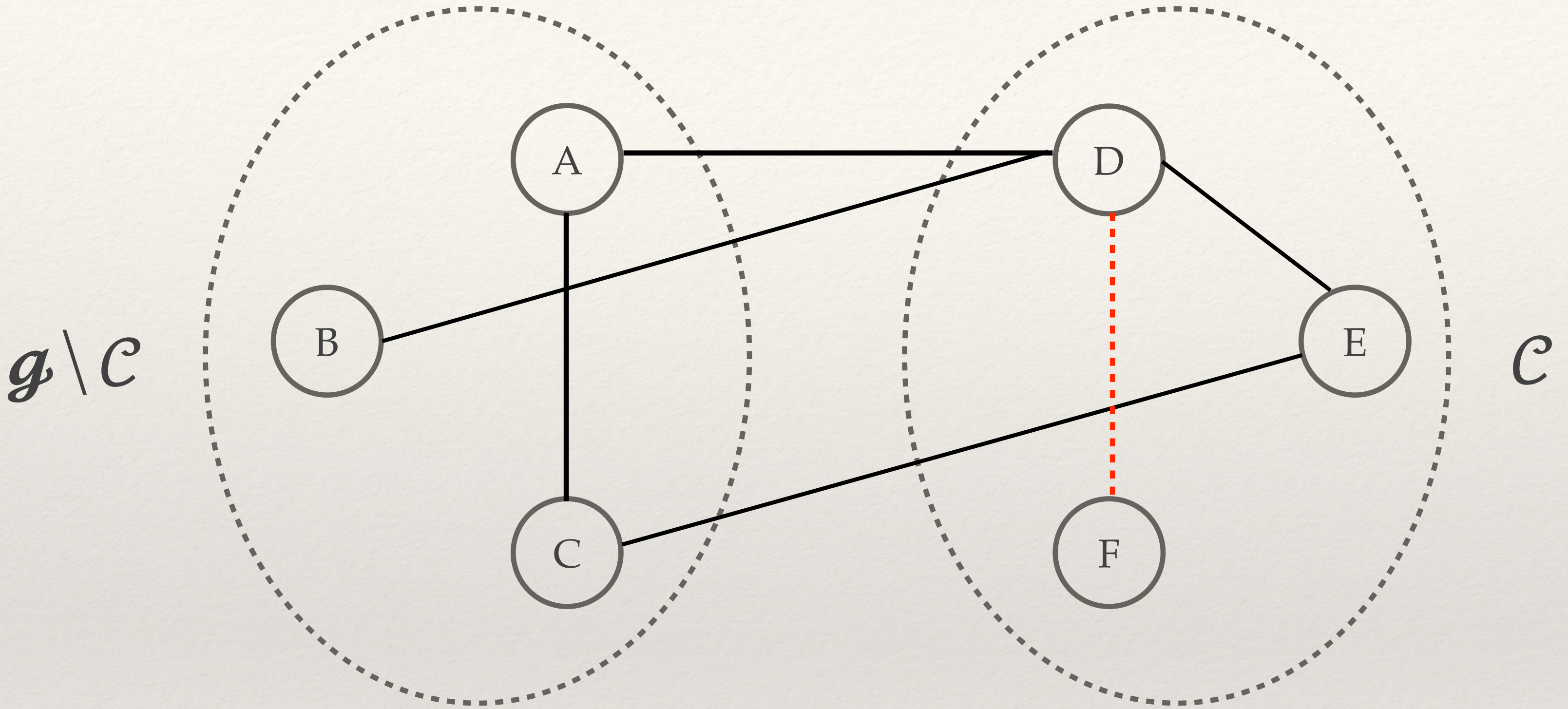
- So far, "Reduce edge weights between partitions"
- Numerator: Cut size of $\mathcal{C}$ from $\mathcal{g} \setminus$ C
- Conductance $\Phi(\mathcal{C}) = \dfrac{c(\mathcal{C}, \mathcal{G} \setminus \mathcal{C})}{\min(k_{\mathcal{C}}, k_{\mathcal{G} \setminus \mathcal{C}})}$

  Denominator: min(total deg in $\mathcal{C}$, total deg in $\mathcal{g} \setminus \mathcal{C}$)
- Ratio cut $\Phi_C(\mathcal{C}) = \dfrac{c(\mathcal{C}, \mathcal{G} \setminus \mathcal{C})}{n_{\mathcal{C}} n_{\mathcal{G} \setminus \mathcal{C}}}$

  Denominator: (# vertices in $\mathcal{C}$) * (# vertices in $\mathcal{g} \setminus \mathcal{C}$)
- Normalized cut $\Phi_N(\mathcal{C}) = \dfrac{c(\mathcal{C}, \mathcal{G} \setminus \mathcal{C})}{k_{\mathcal{C}}}$

  Denominator: Total degree of C

Other measures (Example)



Conductance = 3/min(4,4) = 0.75
Ratio cut = 3/(3*3) = 3
Normalized cut = 3/4 = 0.75

Conductance = 3/min(6,4) = 0.75
Ratio cut = 3/(3*3) = 3
Normalized cut = 3/6 = 0.5

$$\Phi(\mathcal{C}) = \frac{c(\mathcal{C}, \mathcal{G} \setminus \mathcal{C})}{\min(k_{\mathcal{C}}, k_{\mathcal{G}\setminus\mathcal{C}})}$$

$$\Phi_C(\mathcal{C}) = \frac{c(\mathcal{C}, \mathcal{G} \setminus \mathcal{C})}{n_{\mathcal{C}} n_{\mathcal{G}\setminus\mathcal{C}}}$$

$$\Phi_N(\mathcal{C}) = \frac{c(\mathcal{C}, \mathcal{G} \setminus \mathcal{C})}{k_{\mathcal{C}}}$$

Weakness of Graph Partitioning

- Strong assumptions
  - Need to know # groups
  - May even need to know size of groups
- Iterative bisectioning into ≥2 partitions not reliable

Hierarchical Clustering

- Agglomerative algorithms (Bottom-up)
  - Start with vertices, remove all edges
  - Iteratively merge vertices by adding edges
- Divisive algorithms (Top-down)
  - Start with entire graph
  - Iteratively split into partitions by removing edges

Agglomerative algorithms

- Clusters merged based on similarity
- Compute <u>similarity measure</u> between vertices, no matter if they are connected or not
- For edge x_{ij} where i and j are in different clusters,
  - Single linkage clustering [Pick edge x_{ij} with min. weight]
  - Complete linkage clustering [Pick edge x_{ij} with max. weight]
  - Average linkage clustering [Pick edge x_{ij} with avg. weight]

Dendrogram

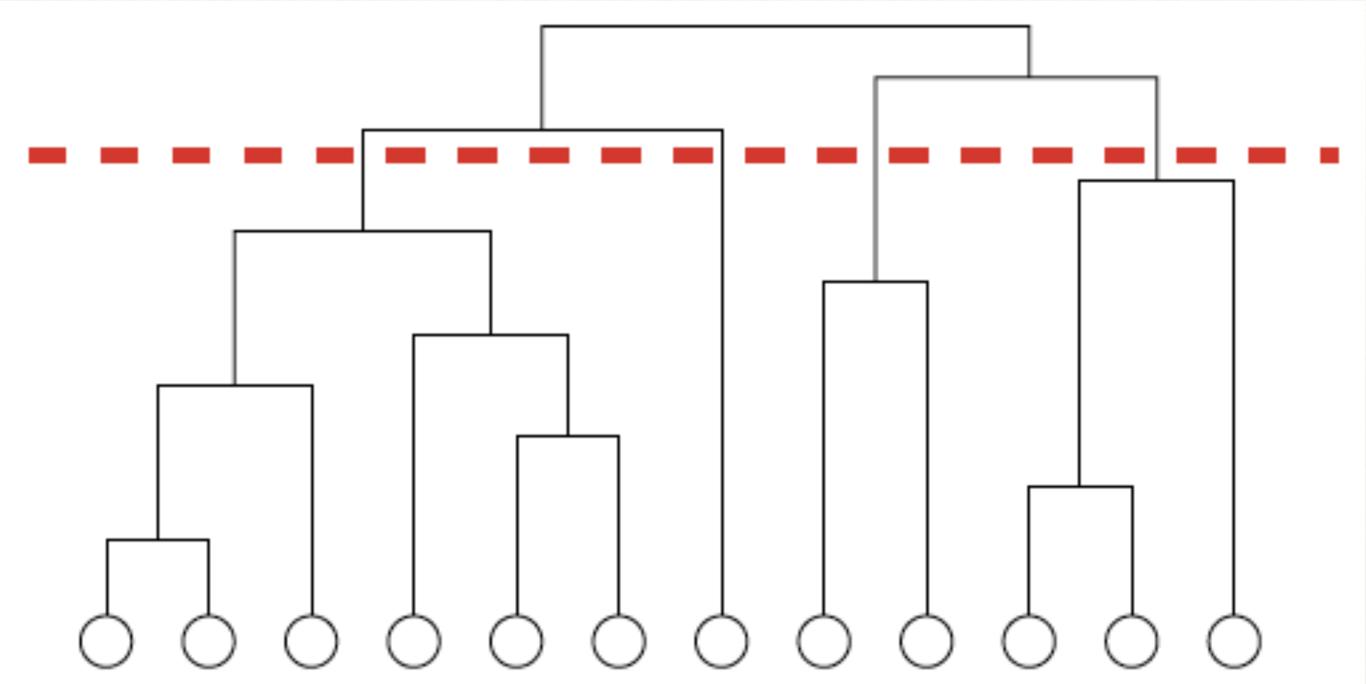


Fig. 8. A dendrogram, or hierarchical tree. Horizontal cuts correspond to partitions of the graph in communities. Reprinted figure with permission from Ref. [54].

Divisive algorithms

- Covered in later section

Pros and Cons of Hierarchical Clustering

- Pros
  - Don't need to assume number or size of clusters
- Cons
  - No way to decide which partition best represents of the community structure in the graph
  - Does not scale well
    (Cost to calculate pairwise similarity measure increases quickly with # vertices)

Partitional Clustering

- Fix # clusters **K**
- Define distance function **d** on metric space **(X,d)**
- Form **K** groups based on distance function
- Examples
  - **Minimum k-clustering** [minimize {largest diameter, among clusters}]
  - **k-clustering sum** [minimize {avg dist between all pairs, among clusters}]
  - **k-center** [min. {max d_i of distances from points to reference point x_i}]
  - **k-median** [Same as k-center. replace max with avg]
  - **k-means clustering**

k-means clustering

- Cost function $\sum_{i=1}^{k} \sum_{\mathbf{x_j} \in S_i} \|\mathbf{x_j} - \mathbf{c_i}\|^2$
- k = #clusters; Centroid c_i ; S_i = Points in i^{th} cluster
- According to paper: Solved using Lloyd's algorithm
  "However, Lloyd's algorithm differs from *k*-means clustering in that its input is a continuous geometric region rather than a discrete set of points."
  [Source: http://en.wikipedia.org/wiki/Lloyd's_algorithm]
- My first intuition: EM algorithm
  "The algorithm as just described monotonically approaches a local minimum of the cost function, and is commonly called *hard EM*. The *k*-means algorithm is an example of this class of algorithms."
  [Source: http://en.wikipedia.org/wiki/Expectation–maximization_algorithm]
- Find out more on Wikipedia:
  http://en.wikipedia.org/wiki/K-means_clustering
- Extensions: Fuzzy k-means

Weakness of Partitional Clustering

- Need to fix #clusters **K**
    - (Thought: What about binary search…?)
- Embedding in metric space may not be natural for the problem at hand

Spectral Clustering

- Didn't really read in-depth
- "Summary"
  - The bulk of Page 95 explains "Why Laplacian matrix is suitable for spectral clustering"
  - Top half of Page 96 draws relation/similarities between spectral clustering and other methods like "graph partitioning" and "random walks"
  - The last paragraph before "5. Divisive algorithms" evaluates the spectral methods

Outline

- Introduction and Definitions
- Approaches
  - Traditional methods
  - **Divisive algorithms**

Divisive algorithms

- Agglomerative algorithms iteratively add edges;
Divisive algorithms iteratively remove edges
- **Algorithm of Girvan and Newman**
  - Historically important - “Marked beginning of a new era in the field of community detection”
  - Based on “edge centrality”
[Estimates importance of edges according to some property / process running on the graph]

Girvan and Newman

- Algorithm (Pg 97)
  - Compute centrality of all edges
  - Removal edge with largest centrality (random if ties)
  - Repeat Step 1 on new graph

Girvan and Newman

- Property used to Girvan and Newman: “Betweenness”
  [Expresses frequency of participation of edges to a process]
- 3 definitions
  - Geodesic edge betweenness
  - Random-walk edge betweenness
  - Current-flow edge betweenness

Geodesic edge betweenness

- Number of shortest paths between all vertex pairs that run along the given edge
- Intuition: “Intercommunity edges have large value of edge betweenness”
- Can be calculated in $O(mn)$, or $O(n^2)$ on sparse graph, with techniques based on BFS

Random-walk edge betweenness

- Idea: Information spreads randomly, not always via shortest path
  - Pick 2 pairs of vertices **s** and **t**
  - Walker moves from s to t, crossing edges with equal probability
  - Compute probability that each edge was crossed by walker
  - Might want to compute "net crossing probability"
    [To negate back/forth walking due to randomness which doesn't say anything about centrality]
    - Fix direction→; Then use | → - ← |
  - Repeat whole process for K pairs of **s** and **t**, then average the edge probabilities

Random-walk edge betweenness

- Wikipedia

Formally, the random walk betweenness centrality of a node is

$$C_i^{RWB} = \sum_{j \neq i \neq k} r_{jk}$$

where the r_{jk} element of matrix R contains the probability of a random walk starting at node j with absorbing node k, passing through node i.

Calculating random walk betweenness in large networks is computationally very intensive.[5]

http://en.wikipedia.org/wiki/Random_walk_closeness_centrality

Current-flow edge betweenness

- Idea: Consider graph as a resistor network, with edges having unit resistance
  - Calculate current carried by each edge by applying a voltage difference between all possible vertex pairs
  - Can be calculated by solving Kirchoff's equations
  - Possible to show that this measure is equivalent to random-walk betweenness

Evaluation

- Calculating Geodesic edge betweenness is much faster than Random-walk and Current-flow edge betweenness
- Numerical studies showed that recalculation of centrality after removing an edge is essential for the Girvan-Newman method
  [i.e. don't just calculate once and keep using the same centrality values]

Modifications to the Girvan-Newman method

- Tyler et al.
  [Calculate contribution to edge betweenness only from a limited number of centers, chosen at random, deriving a sort of Monte Carlo estimate]
  [Feature: Allows overlaps in communities]
- Rattigan et al.
  [Quick approximation of edge betweenness values carried out by using a network structure index]
- Chen and Yuan
  [Count only non-redundant paths]
- Holme et al.
  [Remove vertices rather than edges]

Speed-ups

- Pinney and Westhead
- Gregory
  [CONGA (Cluster Overlap Newman-Girvan Algorithm). Code available]

Allow overlapping communities

Other divisive algorithms

- Idea 1: Inter-cluster edges are related to presence of cycles
  - **Edge clustering coefficient**
    (#triangles, including vertex)/(#possible triangles that can be formed)
  - Remove edge with lowest coefficient. Recalculate. Repeat.

Other divisive algorithms

- Idea 2: Efficiency of information travelling on graph
  - **Information centrality**
- Idea 3: Neighbours of a vertex inside community are "close" to each other
  - **Loop coefficient**

Questions?

- Slides will be made available for reference