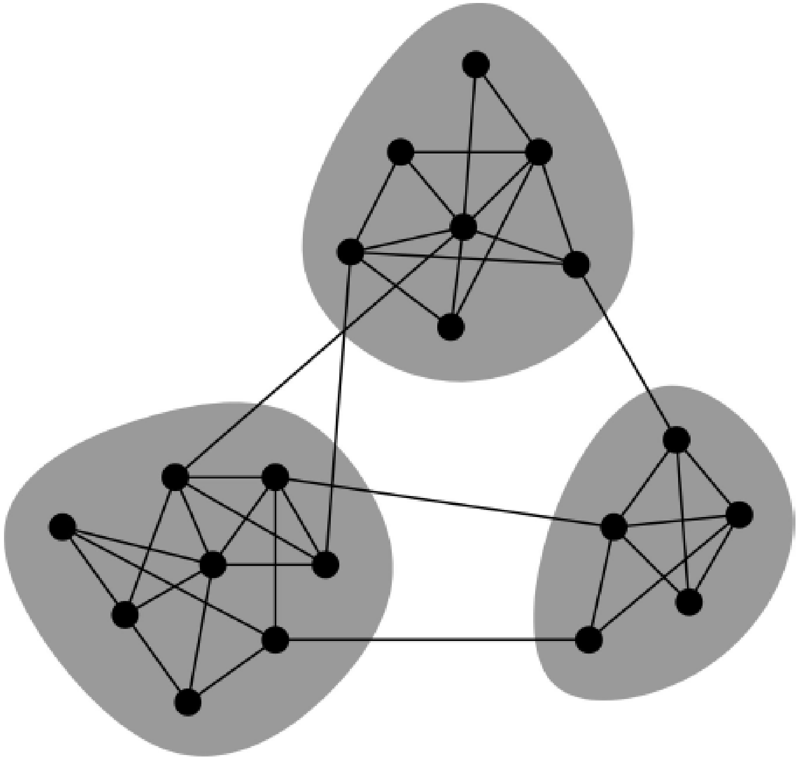
Community Detection: Modularity and Trawling

CS224W: Social and Information Network Analysis
Jure Leskovec, Stanford University
http://cs224w.stanford.edu

Network Communities

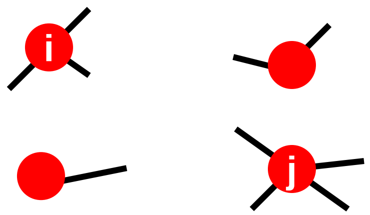
- **Communities:** sets of **tightly connected nodes**
- Define: **Modularity Q**
  - A measure of how well a network is partitioned into communities
  - Given a partitioning of the network into groups $s \in S$:



$$Q \;\propto\; \sum_{s \in S} [\ (\#\text{ edges within group } s) - \underbrace{(\text{expected }\#\text{ edges within group } s)}_{\textbf{Need a null model!}}\ ]$$

Null Model: Configuration Model

- **Given real G, construct rewired network G'**
  - Same degree distribution but random connections
  - Consider G' as multigraph



  - **The expected number of edge between nodes *i* and *j*** of degrees k_i and $k_j = k_i \cdot \frac{k_j}{2m} = \frac{k_i k_j}{2m}$
    - The expected number of edges in (multigraph) G':
      - $= \frac{1}{2}\sum_{i \in N}\sum_{j \in N}\frac{k_i k_j}{2m} = \frac{1}{2} \cdot \frac{1}{2m}\sum_{i \in N} k_i\left(\sum_{j \in N} k_j\right) =$
      - $= \frac{1}{4m} 2m \cdot 2m = m$

Note:
$$\sum_{u \in N} k_u = 2m$$

Modularity

- **Modularity of partitioning C of graph G:**
  - Q ∝ $\sum_{s \in S}$ [(# edges within group *s*) –
    (expected # edges within group *s*)]
  - $Q(G,S) = \underbrace{\frac{1}{2m}}_{}\sum_{s\in S}\sum_{i\in s}\sum_{j\in s}\left(A_{ij} - \frac{k_i k_j}{2m}\right)$

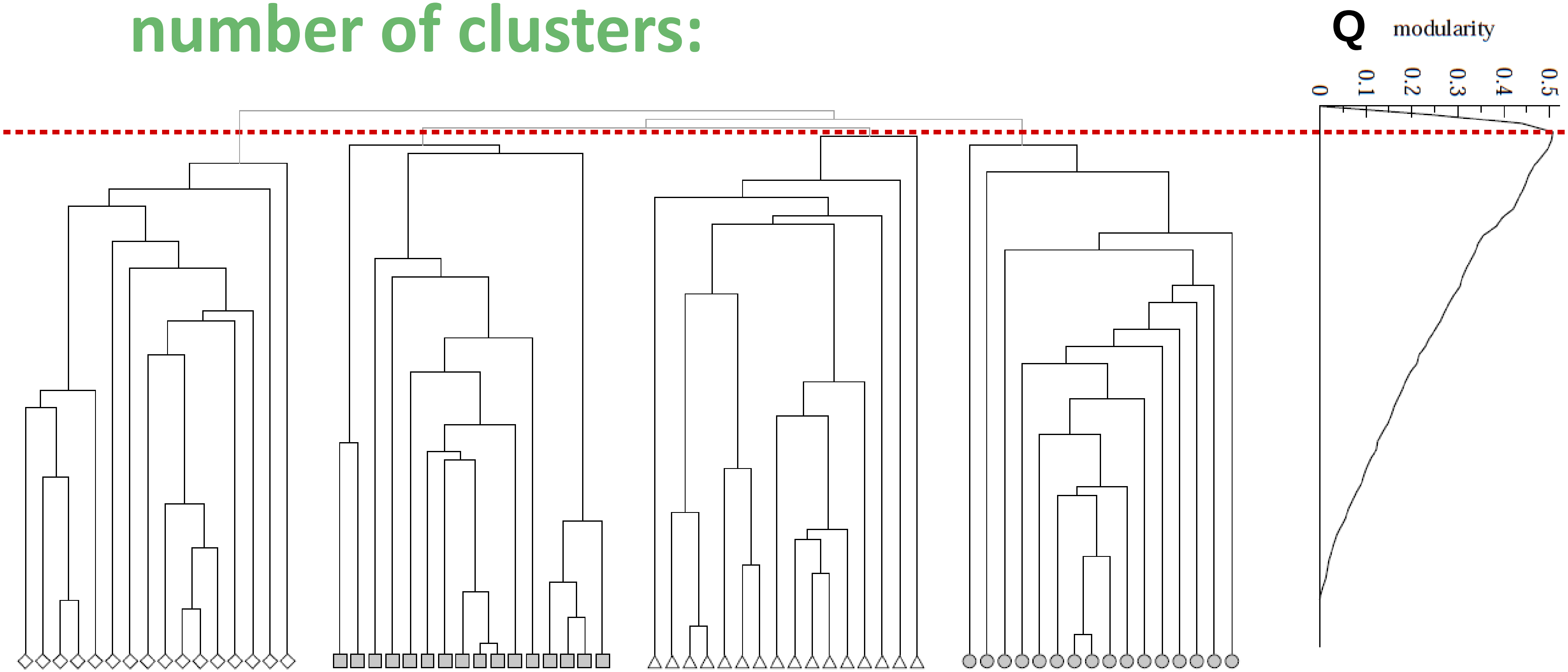
    Normalizing cost.: -1<Q<1

    A_{ij} = 1 if i→j,
    0 else

- **Modularity lies in the range [–1,1]**
  - It is positive if the number of edges within groups exceeds the expected number
  - 0.3<Q<0.7 means significant community structure

Modularity: Number of clusters

- **Modularity is useful for selecting the number of clusters:**



Why not optimize modularity directly?

Method 2: Modularity Optimization

- Let's split the graph into 2 communities
- What to directly optimize modularity!
  - $\max_S Q(G,S) = \frac{1}{2m} \sum_{s \in S} \sum_{i,j \in s} \left(A_{ij} - \frac{k_i k_j}{2m} \right)$
- **Community membership vector s:**
  - s_i = 1 if node i is in community 1
    -1 if node i is in community -1

$\frac{s_i s_j + 1}{2} =$ 1.. if $s_i = s_j$
0.. else

- $Q(G,s) = \frac{1}{4m} \sum_{i,j \in N} \left(A_{ij} - \frac{k_i k_j}{2m} \right) (s_i s_j + 1)$
- $= \frac{1}{4m} \sum_{i,j \in N} \left(A_{ij} - \frac{k_i k_j}{2m} \right) s_i s_j$

Modularity Matrix

- **Define:**
  - Modularity matrix: $B_{ij} = A_{ij} - \frac{k_i k_j}{2m}$
  - Membership: s={-1, +1}

Note: each row /column of B sums to 0

- Then: $Q(G,s) = \frac{1}{4m} \sum_{i,j \in N} B_{ij} s_i s_j$

$$= \frac{1}{4m} \sum_i s_i \sum_j B_{ij} s_j = \frac{1}{4m} s^T B s$$

- **Task:** Find $s \in \{-1,+1\}^n$ that maximizes Q(G,s)

- Rewrite *Q* in terms of eigenvalues β_i and eigenvectors u_i of modularity matrix *B*

Modularity Optimization

- **Rewrite:** $Q(G,s) = \frac{1}{4m} s^{\mathrm{T}} B s$

$$= \mathrm{s}^{\mathrm{T}} \left[\sum_{i=1}^{n} u_i \beta_i u_i^T \right] s = \sum_{i=1}^{n} s^T u_i \beta_i u_i^T s$$

$$\boxed{= \sum_{i=1}^{n} (s^T u_i)^2 \beta_i}$$

Note: $\beta_1 > \beta_2 > \cdots$

- If there would be no constraints on *s* then to maximize Q, the easiest way is to make **s** = λ u_1
  - Assigns all weight in the sum to β_1 (largest eigval)
    - All other $\mathbf{s}^{\mathrm{T}}\mathbf{u}_{\mathrm{i}}$ terms zero because of orthonormality
  - But, elements of ***s*** must be $\in${-1,+1}, **NP-hard in general**

Finding Vector s

$$\max_{s} \mathrm{Q}(G, s) = \sum_{i=1}^{n} (s^T u_i)^2 \beta_i \approx \left[\sum_{i=1}^{n} s_i \cdot u_{1,i}\right]^2 \beta_1$$

- **Let's maximize:** $\sum_{i=1}^{n} s_i \cdot u_{1,i}$ where $s_i \in \{-1,+1\}$
- To do this, we set:

$$s_i = \begin{cases} +1 & \text{if } i\text{th element of } \mathbf{u}_1 \geq 0, \\ -1 & \text{if } i\text{th element of } \mathbf{u}_1 < 0. \end{cases}$$

- Similar in spirit to the spectral partitioning algorithm (we will explore this next time)
- Continue the bisection hierarchically

Summary: Modularity Optimization

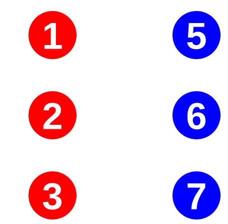
- **Fast Modularity Optimization Algorithm:**
  - Find leading eigenvector u_1 of modularity matrix B
  - Divide the nodes by the signs of the elements of u_1
  - Repeat hierarchically until:
    - If a proposed split does not cause modularity to increase, declare community indivisible and do not split it
    - If all communities are indivisible, stop
- **How to find u_1? Power method!**
  - Start with random $v^{(1)}$, repeat :
  - When converged ($v^{(t)} \approx v^{(t+1)}$), set $u_1 = v^{(t)}$

$$v^{(t+1)} = \frac{Bv^{(t)}}{\left\| Bv^{(t)} \right\|}$$

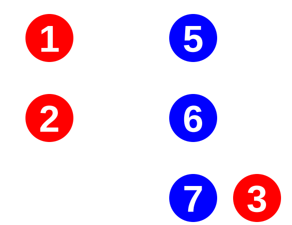
Additional Heuristic Approaches

- **(1) Greedy post-processing:**
  - Start with nodes in two groups, *s*
  - Repeat $t = 1..n$ until all nodes have been moved:
    - For $i = 1..n$
      - Consider moving node i, compute new $Q_t(s_i)$
    - Move node j that hasn't yet been moved and that maximizes $Q_t(s_j)$
      - Note that Q_t can decrease with time t
  - Once iteration is complete, find intermediate state t with highest Q_t
  - Start from this state and repeat until Q stops increasing

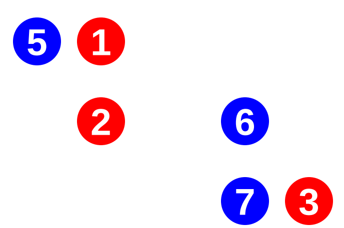
Start:



Move best not-yet-moved node (3), store Q_1



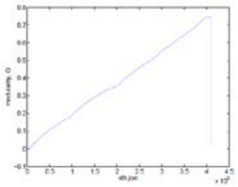
Move best not-yet-moved node (5), store Q_2



Dot this for every not-yet-moved node, pick state x that max Q_t

Additional Heuristic Approaches

- **(2) Clauset-Newman-Moore Algorithm:**
  - **Agglomerative clustering:** start with each node as a separate community, join communities into bigger ones
  - (1) Put each node in its own community x
  - (2) Compute ΔQ_{xy} for all community pairs
  - (3) Merge the pair with largest increase in ΔQ_{xy}
  - Repeat (2)&(3) until only one community remains
- **How to compute $\Delta Q(x,y)$?**
  - Matrix $\Delta Q_{xy} = \frac{1}{2m}\left(1 - \frac{k_i k_j}{2m}\right)$ if node x links y, else $\Delta Q_{xy}=0$
  - If we join communities x and y into a new y, update ΔQ:
    - Remove row/column x of ΔQ
    - For every *k* update: $\Delta Q_{yk} = \Delta Q_{xk} + \Delta Q_{yk}$



Modularity Optimization Methods

network	size n	modularity Q GN	CNM	DA	Fast modularity
karate	34	0.401	0.381	0.419	0.419
jazz musicians	198	0.405	0.439	0.445	0.442
metabolic	453	0.403	0.402	0.434	0.435
email	1133	0.532	0.494	0.574	0.572
key signing	10 680	0.816	0.733	0.846	0.855
physicists	27 519	–	0.668	0.679	0.723

GN = Betweenness centrality, $O(n^3)$
CNM = Clauset-Newman-Moore ($n \log^2 n$)
DA = External pptimization $O(n^2 \log^2 n)$

- **Issues with modularity:**
  - May not find communities with less than $\sqrt{m}$ links
  - NP-hard to optimize exactly [Brandes et al. '07]

Summary: Modularity

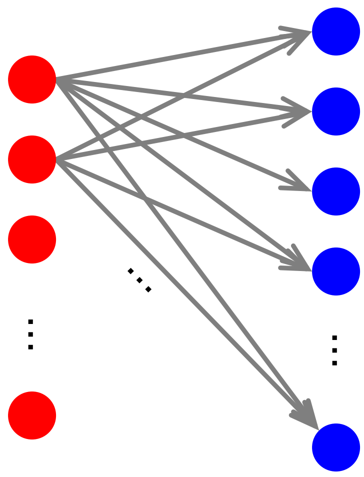
- Girvan-Newman (previous lecture):
  - Based on the "strength of weak ties"
  - Remove edge of highest betweenness
- Modularity:
  - Overall quality of the partitioning of a graph
  - Use to determine the number of communities
- Fast modularity optimization:
  - Transform the modularity optimization to a eigenvalue problem
- Clauset-Newman-Moore:
  - Agglomerative clustering based on Modularity

Trawling for Web Communities

[Kumar et al. '99]

Method3: Trawling

- Searching for small communities in the Web graph
- What is the signature of a community / discussion in a Web graph?



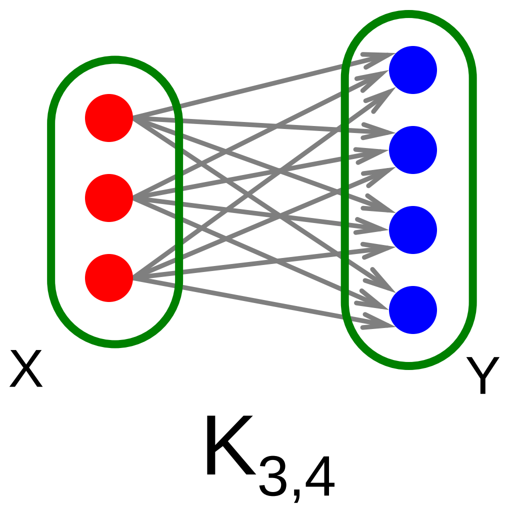
Dense 2-layer graph

Use this to define "topics":
What the same people on the left talk about on the right
Remember HITS!

Intuition: Many people all talking about the same things

Searching for Small Communities

- **A more well-defined problem:**
  Enumerate complete bipartite subgraphs $K_{s,t}$
  - Where $K_{s,t}$: s nodes on the "left" where each links to the same t other nodes on the "right"



$|X| = s = 3$
$|Y| = t = 4$

Fully connected

[Kumar et al. '99]

The Plan: (1), (2) and (3)

- **Two points:**
  - **(1) Dense bipartite graph:** the signature of a community/discussion
  - (2) Complete bipartite subgraph $K_{s,t}$
    - $K_{s,t}$ = graph on s nodes, each links to the same t other nodes
- **Plan:**
  - **(A) From (2) get back to (1):**
    - **Via:** Any dense enough graph contains a smaller $K_{s,t}$ as a subgraph
  - **(B) How do we solve (2) in a giant graph?**
    - What similar problems were solved on big non-graph data?
    - **(3) Frequent itemset enumeration** [Agrawal-Srikant '99]

[Agrawal-Srikant '99]

Frequent Itemset Enumeration

- **Marketbasket analysis:**
  - **What items are bought together in a store?**
- **Setting:**
  - **Market:** Universe U of n items
  - **Baskets:** m subsets of U: $S_1, S_2, \ldots, S_m \subseteq U$ (S_i is a set of items one person bought)
  - **Support:** Frequency threshold f
- **Goal:**
  - **Find all subsets T s.t. $T \subseteq S_i$ of $\geq f$ sets S_i** (items in T were bought together at least f times)

Products sold in a store

Frequent Itemsets: Example

- **Given:**
  - **Universe of items:**
    - U={1,2,3,4,5}
  - **Market baskets:**
    - S_1={1,3,5}, S_2={2,3,4}, S_3={2,4,5}, S_4={3,4,5}, S_5={1,3,4,5}, S_6={2,3,4,5}
  - **Minimum support:** f = 3
  - **Goal:** Find all sets T that appear in at least f S_i's
    - Call such itemsets T **frequent itemsets** (they have support ≥f)
- **Algorithm: Build the lists bottom-up**
  - **Insight:** For a frequent set of size *k*, all its subsets are also frequent

Support of T={2,3} is 2 (T appears in S_2 and S_6)

If T={3,4,5} is frequent, then {3,4}, {3,5}, {4,5} must also be frequent!

[Agrawal-Srikant ‘99]

Example: the Apriori Algorithm

- **Setting:**
  - $U=\{1,2,3,4,5\}$, $f=3$
  - $S_1=\{1,3,5\}$, $S_2=\{2,3,4\}$, $S_3=\{2,4,5\}$, $S_4=\{3,4,5\}$, $S_5=\{1,3,4,5\}$, $S_6=\{2,3,4,5\}$

Itemset size	Itemsets
1	~~{1}~~ {2} {3} {4} {5}
2	~~{2, 3}~~ {2, 4} ~~{2, 5}~~ {3, 4} {3, 5} {4, 5}
3	~~{2, 3, 4}~~ {3, 4, 5}
4	{}

2 steps:
1) Candidate generation
2) Pruning

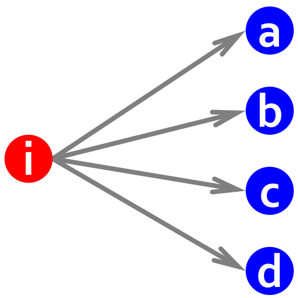
[Agrawal-Srikant '99]

The Apriori Algorithm

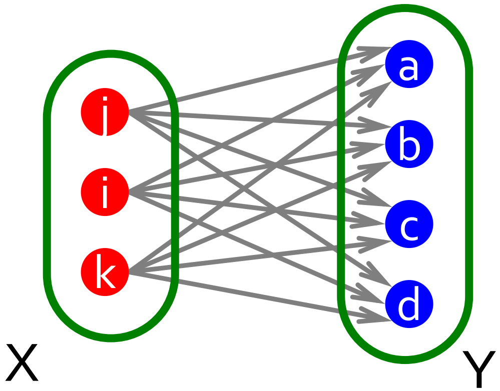
- **For $i = 1, \ldots, k$**
  - Generate all sets of size i by composing sets of size i-1 that differ in 1 element
  - Prune the sets of size i with support < f
- **Open question:**
  - Efficiently find only maximal frequent sets
- **What's the connection between itemsets and complete bipartite graphs?**

From Itemsets to Bipartite $K_{s,t}$

- **Itemsets finds Complete bipartite graphs**
- **How?**
  - View each node i as a set S_i of nodes i points to
  - $K_{s,t}$ = a set Y of size t that occurs in s sets S_i
  - Looking for $K_{s,t}$ → set of frequency threshold to s and look at layer t – all frequent sets of size t



S_i={a,b,c,d}



s … minimum support (|X|=s)
t … itemset size

From $K_{s,t}$ to Communities

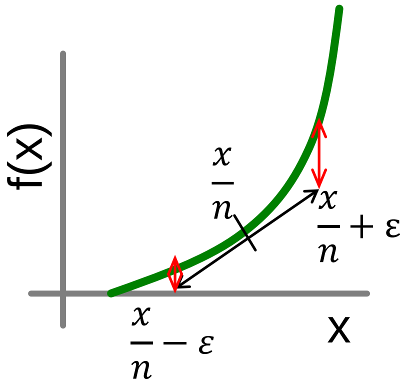
- **From $K_{s,t}$ to Communities:** Informally, every dense enough graph G contains a bipartite subgraph $K_{s,t}$ where s and t depend on size (# of nodes) and density (avg. degree) of G
  [Kovan-Sos-Turan '53]

- **Theorem:**
  Let $G=(X,Y,E)$, $|X|=|Y|=n$
  with avg. degree $\overline{k} = s^{\frac{1}{t}}\, n^{1-\frac{1}{t}} + t$
  then G contains $K_{s,t}$ as a subgraph.

Proof: $K_{s,t}$ and Communities

For the proof we will need the following fact

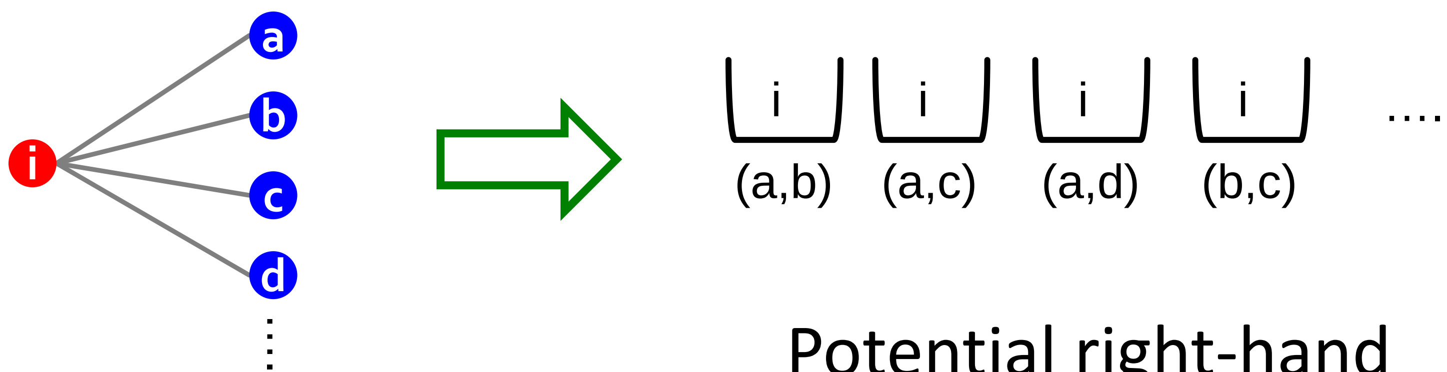
- Recall: $\binom{a}{b} = \frac{a(a-1)\dots(a-b+1)}{b!}$
  - Let $f(x) = x(x-1)(x-2)\dots(x-k)$
    Once $x \geq k$, $f(x)$ curves upward (convex)



- **Suppose a setting:**
  - $g(y)$ is convex
  - Want to minimize $\sum_{i=1}^{n} g(x_i)$
  - where $\sum_{i=1}^{n} x_i = x$
- **To minimize $\sum_{i=1}^{n} g(x_i)$ make each $x_i = \frac{x}{n}$**

Nodes and Buckets

- Consider node i of degree k_i and neighbor set S_i



- Put node i in buckets for all size t subsets of i's neighbors

Potential right-hand sides of $K_{s,t}$ (*i.e.*, all size t subsets of S_i)

As soon as s nodes appear in a bucket we have a $K_{s,t}$

Nodes and Buckets

- Note: As soon as s nodes appear in a bucket we found a $K_{s,t}$
- **How many buckets does node *i* contribute to?**

$$\binom{k_i}{t}$$

= # of ways to select t elements out of k_i

k_i … degree of node i

- What is the total size of all buckets?

$$\sum_{i=1}^{n}\binom{k_i}{t} \geq \sum_{i=1}^{n}\binom{\bar{k}}{t} = n\binom{\bar{k}}{t}$$

By convexity
($k_i > t$)

$$\bar{k} = \frac{1}{n}\sum_{i \in N} k_i$$

Nodes and Buckets

$$\binom{a}{b} = \frac{a(a-1)\ldots(a-b+1)}{b!}$$

- So, the total height of all buckets is…

$$n\binom{\bar{k}}{t} \geq n\frac{\left(\bar{k}-t\right)^{t}}{t!} = n\frac{\left(s^{\frac{1}{t}}\, n^{1-\frac{1}{t}} + t - t\right)^{t}}{t!}$$

$$= \frac{n\, s\, n^{t-1}}{t!} = \frac{n^{t}\, s}{t!}$$

Plug in:

$$\bar{k} = s^{\frac{1}{t}}\, n^{1-\frac{1}{t}} + t$$

And We are Done!

- We have: Total height of all buckets: $\geq \frac{n^t s}{t!}$
- How many buckets are there? $\binom{n}{t} \leq \frac{n^t}{t!}$
- What is the average height of buckets?

$$\geq \frac{n^t s}{t!} \frac{t!}{n^t} = s$$

So, avg. bucket height $\geq$ s

- $\Rightarrow$ By pigeonhole principle, there must be at least one bucket with more than s nodes in it.
- $\Rightarrow$ We found a $K_{s,t}$

[Kumar et al. '99]

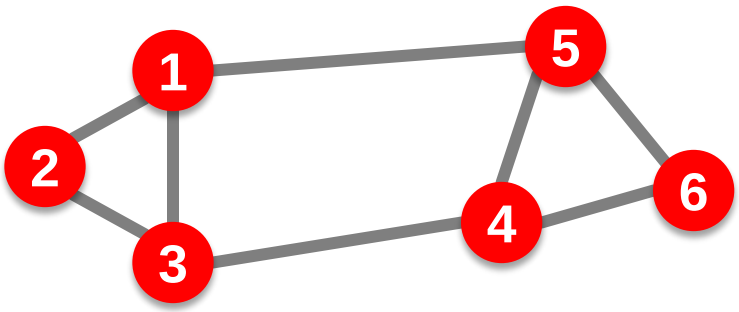
Method3: Trawling — Summary

- **Analytical result:**
  - Complete bipartite subgraphs $K_{s,t}$ are embedded in larger dense enough graphs (*i.e.*, the communities)
    - Biparite subgraphs act as "signatures" of communities
- **Algorithmic result:**
  - Frequent itemset extraction and dynamic programming finds graphs $K_{s,t}$
  - **Method is super scalable**

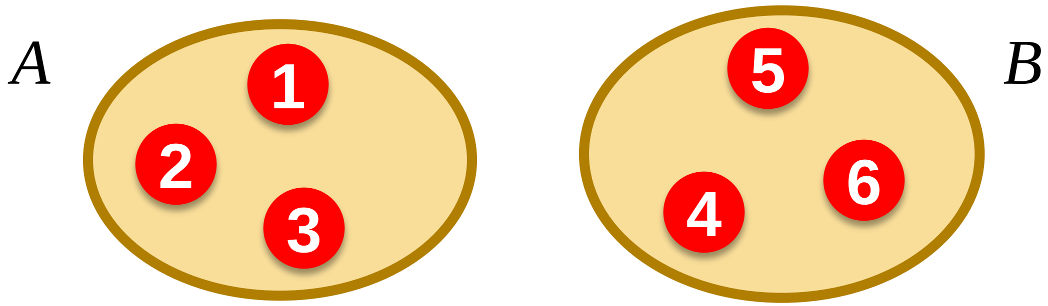
Spectral Graph Partitioning

Method4: Graph Partitioning

- **Undirected graph G(V,E):**



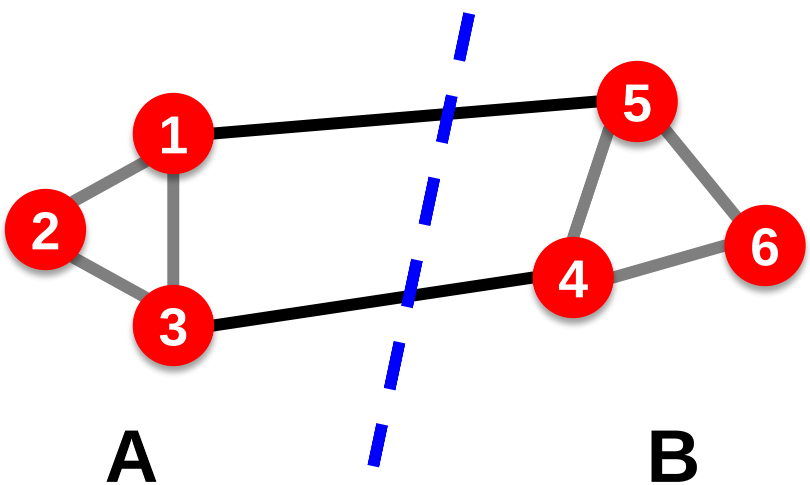
- **Bi-partitioning task:**
  - Divide vertices into two disjoint groups (A,B)



- **Questions:**
  - How can we define a "good" partition of G?
  - How can we efficiently identify such a partition?

Graph Partitioning

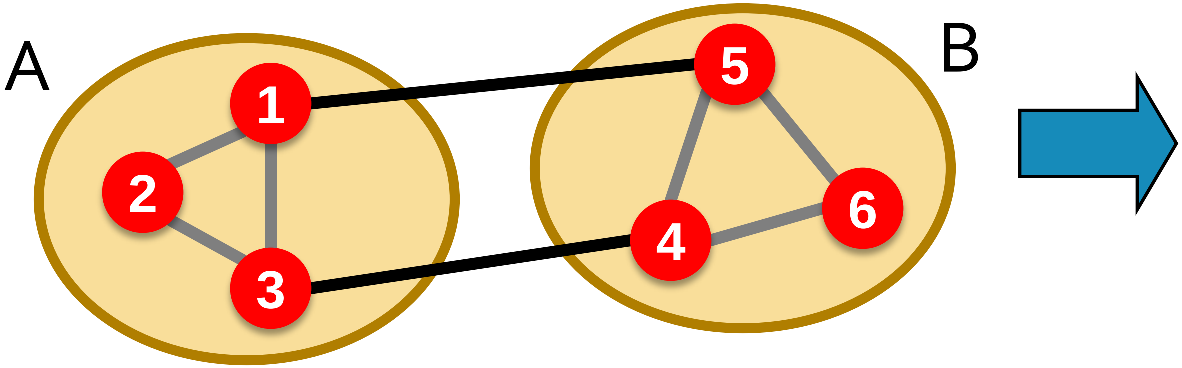
- **What makes a good partition?**
  - Maximize the number of within-group connections
  - Minimize the number of between-group connections



Graph Cuts

- **Express partitioning objectives as a function of the "edge cut" of the partition**
- Cut: Set of edges with only one vertex in a group:

$$cut(A,B) = \sum_{i \in A, j \in B} w_{ij}$$



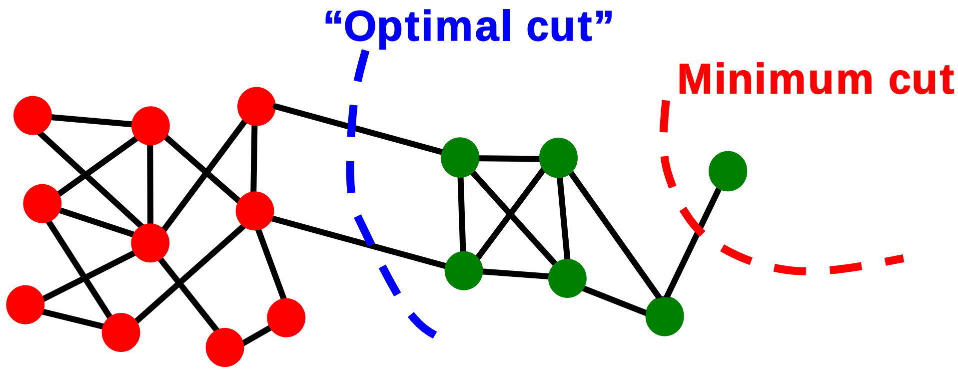
$cut(A,B) = 2$

Graph Cut Criterion

- **Criterion: Minimum-cut**
  - Minimise weight of connections between groups

$$\min_{A,B} \; cut(A,B)$$

- **Degenerate case:**



- **Problem:**
  - Only considers external cluster connections
  - Does not consider internal cluster connectivity

Graph Cut Criteria

- **Criterion:** **Normalized-cut** [Shi-Malik, '97]
  - Connectivity between groups relative to the density of each group

$$ncut(A,B) = \frac{cut(A,B)}{vol(A)} + \frac{cut(A,B)}{vol(B)}$$

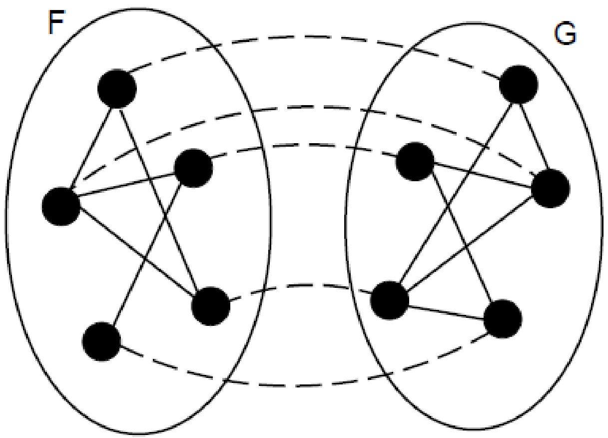
vol(A): total weight of the edges with at least one endpoint in A: $\mathrm{vol}(A) = \sum_{i \in A} k_i$

- **Why use this criterion?**
  - Produces more balanced partitions
- **How do we efficiently find a good partition?**
  - **Problem:** Computing optimal cut is NP-hard

Competition Results: Graph Alignment

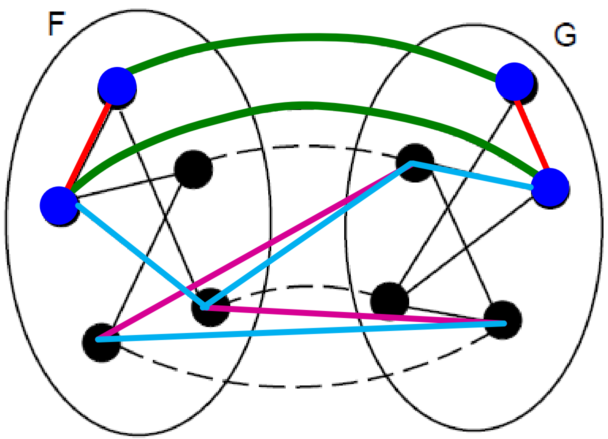
Wikipedia Graph Alignment

- Given the **G**erman and **F**rench Wikipedia graph
- And a few example corresponding articles



- **Goal:** Find the remaining correspondences:
  - Link "Paris" in German to "Paris" in French
    - Intuition: Paris in both languages links to "similar" pages (pages that also link to each other)

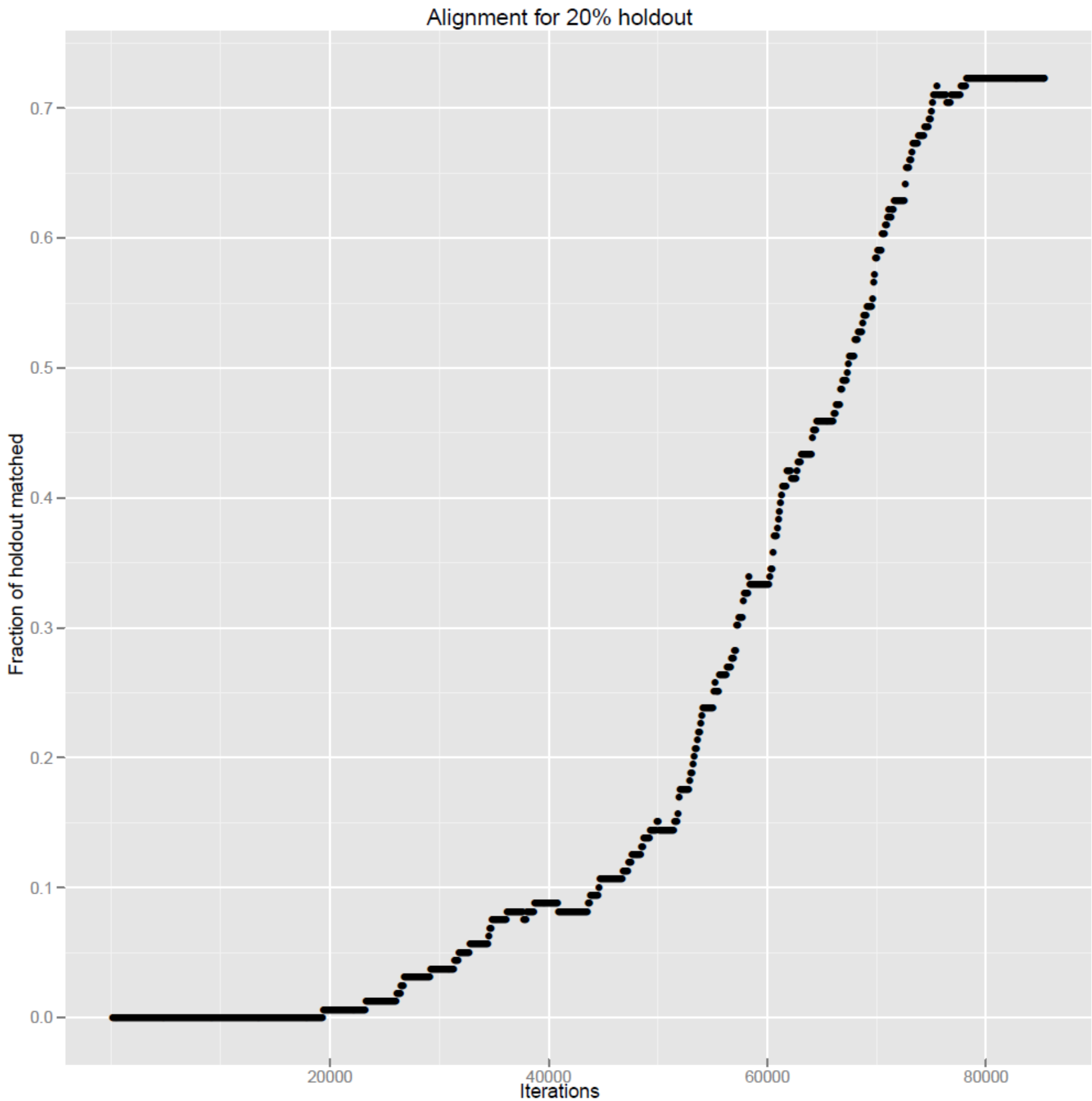
Approach 1: Square Maximization



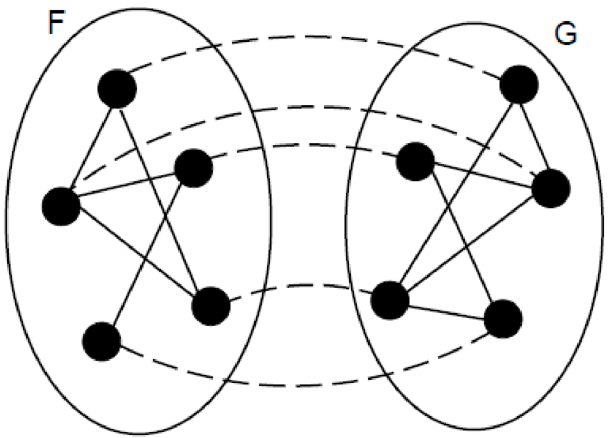
Winning solution:

- Start from some pairing S
  - Start from random pairing
- Goodness of pairing S:
  - Number of "squares"
- Consider transforming $(u_F, u_G), (v_F, v_G)$ to $(v_F, u_G), (u_F, v_G)$
- Accept the swap if the number of squares increases
- **Improvements:**
  - Bound on swap improvement:
    - No need to swap nodes that don't give good improvement
  - Computing swap change efficiently

Approach 1: Square Maximization



Approach 2: Machine Learning



- For a pair of nodes (u_F, u_G) construct a feature vector
  - Matches from the training set (M.txt) are "positive" examples
  - Pairs not in M.txt are "negative" examples
- Use Random Forests to label pairs (AUC=0.87)
  - Each pair gets a probability that they match
- Now greedily fill-in the remaining pairings by considering correspondence probabilities

Results and Extra Credit

ID	# Correct	Fraction
krish (10%)	3,308	0.83
pmk (8%)	2,941	0.74
lussier1 (6%)	2,191	0.55
prgao (4%)	2,107	0.53
jieyang (4%)	1,706	0.43
carmenv	978	0.24
anmittal	861	0.22
adotey	828	0.21
billyue	805	0.20
gibbons4	507	0.13
leonlin	145	0.04
cktan	65	0.02