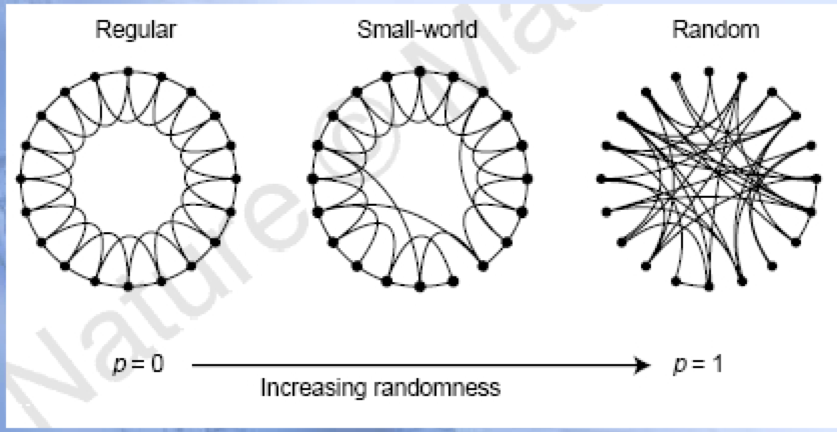
Community Detection in Social Networks

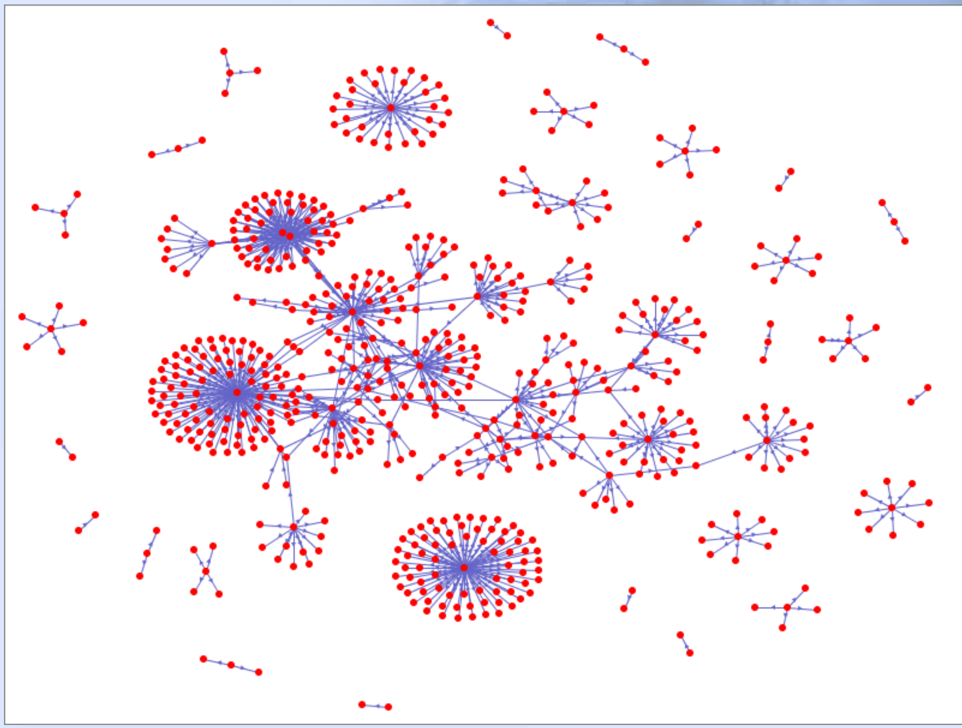
Lei Tang

Properties of Complex Network

Power Law



Small World

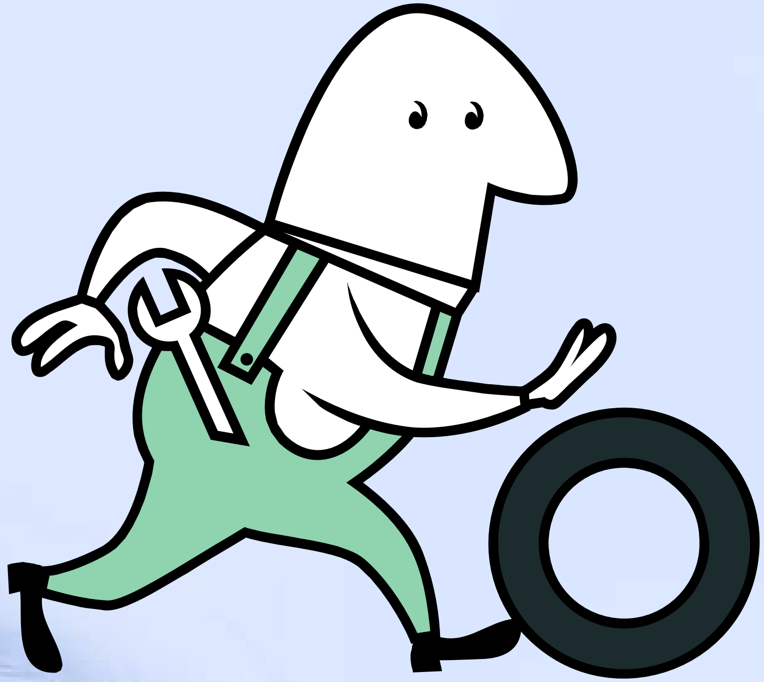


Community Structure

Why Community Detection?

- Communities in a citation network might represent related papers on a single topic;
- Communities on the web might represent pages of related topics;
- Community can be considered as a summary of the whole network thus easy to visualize and understand.
- Sometimes, community can reveal the properties without releasing the individual privacy information.

Community Detection, Reinventing the wheel?



Community Detection = Clustering?

- As I understand, community detection is essentially clustering.
- But why so many works on Community Detection? (in physical review, KDD, WWW)
- The network data pose challenges to classical clustering method.

Difference

- Clustering works on the distance or similarity matrix (k-means, hierarchical clustering, spectral clustering)
- Network data tends to be “discrete”, leading to algorithms using the graph property directly (k-clique, quasi-clique, vertex-betweenness, edge-betweeness etc.)
- Real-world network is large scale! Sometimes, even n^2 in unbearable for efficiency or space (local/distributed clustering, network approximation, sampling method)

Outline

- Two recent community detection methods
- Clustering based on shortest-path betweenness
- Clustering based on network modularity

Basic Idea

- *A simple divisive strategy:*
- *Repeat*

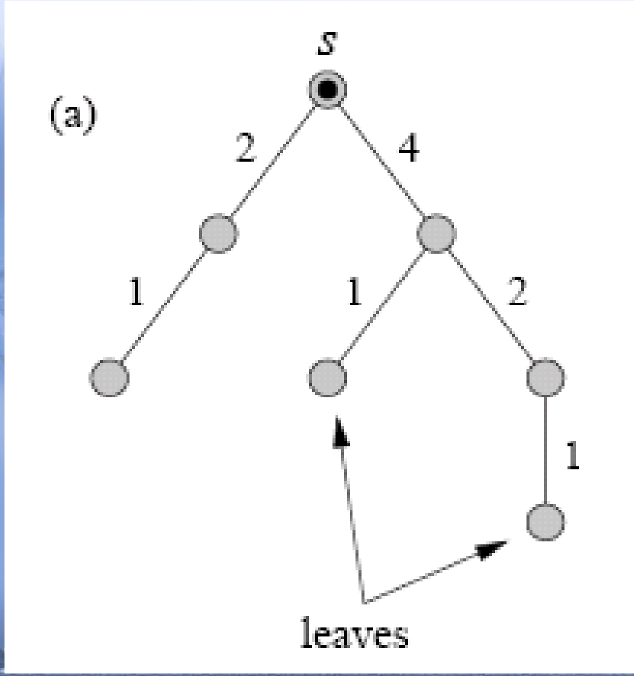
1. *Find out one “inter-community” edge*
2. *Remove the edge*
3. *Check if there’s any disconnected components (which corresponds to a community)*

How to measure “inter-community”

- *If two communities are joined by a few inter-community edges, then all the paths from one community to another must pass the edges.*
- Various measures:
- Edge Betweenness: find the shortest paths between all pairs of nodes and count how many run along each edge.
- Random Walk betweenness.
- Current-flow betweenness

Shortest-path betweenness

- Computation could be expensive: calculating the shortest path between one pair is O(m), and there are O(n^2) pairs.
- Could be optimized to O(mn)
- Simple case: only one shortest path



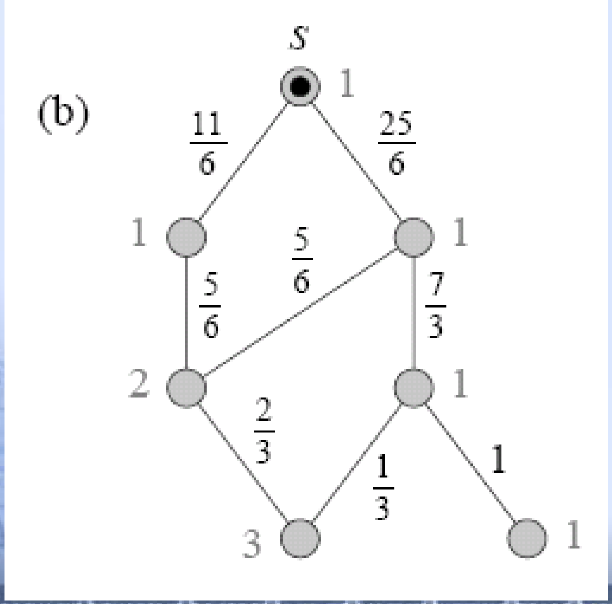
When there is only one single path between the Source S and other vertex, then those paths form a tree.

Bottom-up: start from the leaves, assign edges to 1.

Count of parent edge = sum (count of children edge)+1

Multiple shortest path

- First compute the number of paths from source to other vertex
- Then assign a proper weight for the path counts
- sum of the betweenness =.number of reachable vertices.



Calculate #shortest path

1.Initial distance $d_s = 0$ $w_s = 1$

2. Every vertex i adjacent to s is given distance $d_i = d_s + 1 = 1$, and weight $w_i = w_s = 1$.

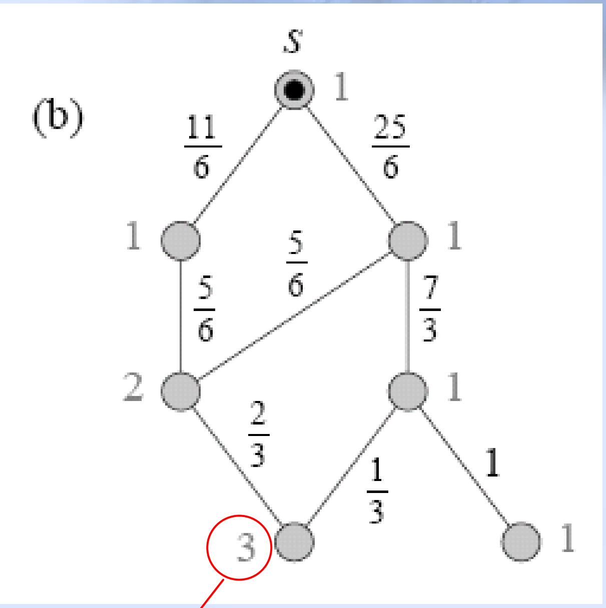
3. For each vertex j adjacent to one of *those* vertices i we do one of three things:

   (a) If j has not yet been assigned a distance, it is assigned distance $d_j = d_i + 1$ and weight $w_j = w_i$.

   (b) If j has already been assigned a distance and $d_j = d_i + 1$, then the vertex's weight is increased by w_i, that is $w_j \leftarrow w_j + w_i$.

   (c) If j has already been assigned a distance and $d_j < d_i + 1$, we do nothing.

4. Repeat from step 3 until no vertices remain that have assigned distances but whose neighbors do not have assigned distances.



W:Number of shortest paths

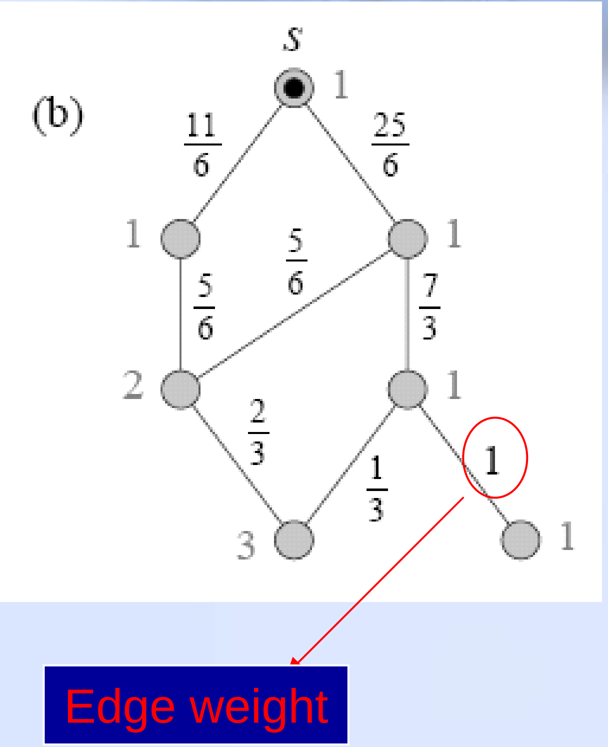
Update edge weight

1. Find every "leaf" vertex t, i.e., a vertex such that no paths from s to other vertices go though t.

2. For each vertex i neighboring t assign a score to the edge from t to i of w_i/w_t.

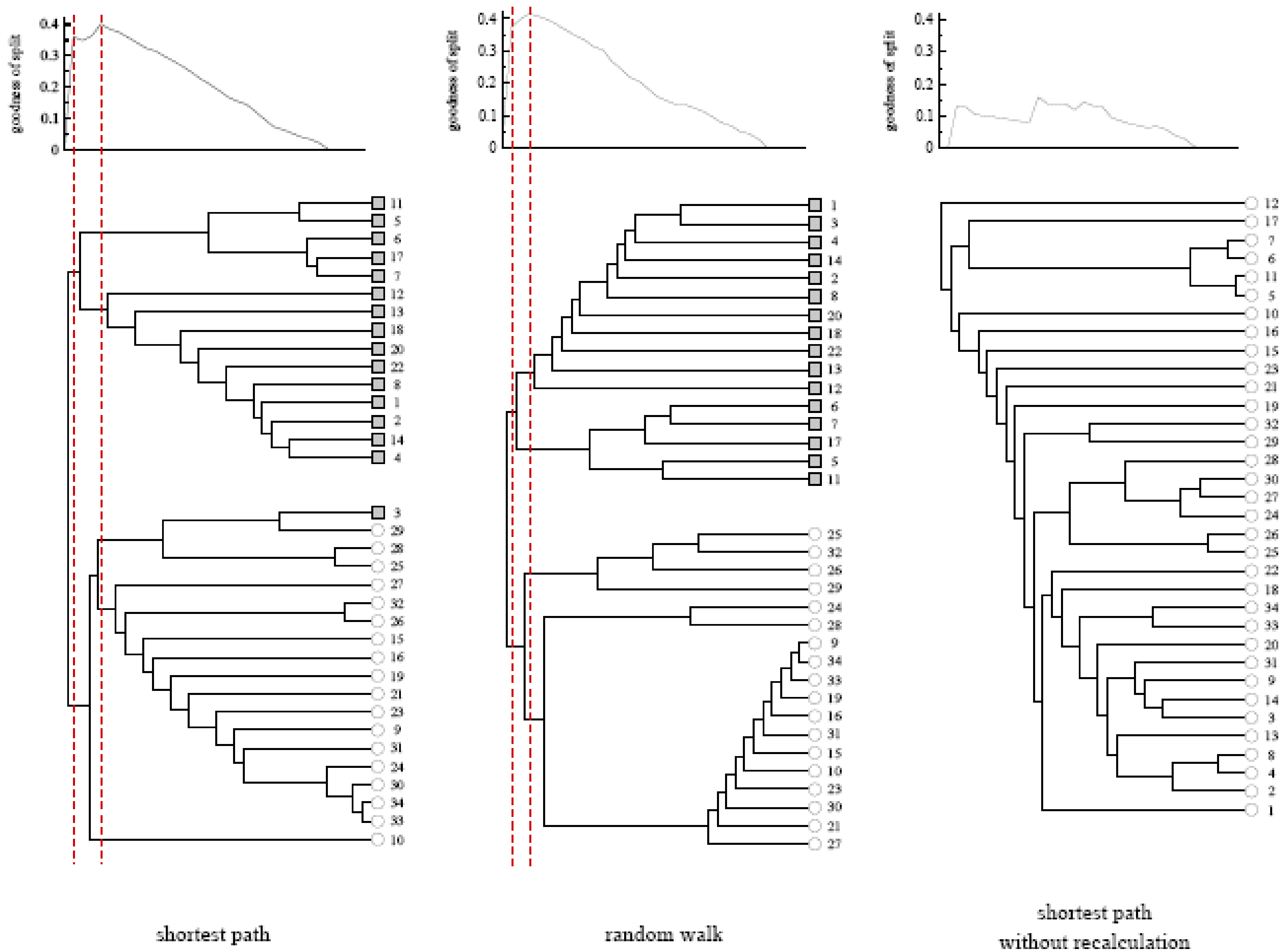
3. Now, starting with the edges that are farthest from the source vertex s—lower down in a diagram such as Fig. 4b—work up towards s. To the edge from vertex i to vertex j, with j being farther from s than i, assign a score that is 1 plus the sum of the scores on the neighboring edges immediately below it (i.e., those with which it shares a common vertex), all multiplied by w_i/w_j.

4. Repeat from step 3 until vertex s is reached.



Time Complexity

- O(mn) in each iteration.
- Could be accelerated by noting that only the nodes in the connected component would be affected.
- Some other techniques developed: sampling strategy to approximate the betweenness; use specific network index for speed.


goodness of split
0.4
0.3
0.2
0.1
0
11
5
6
17
7
12
13
18
20
22
8
1
2
14
4
3
29
28
25
27
32
26
15
16
19
21
23
9
31
24
30
34
33
10
shortest path
goodness of split
0.4
0.3
0.2
0.1
0
1
3
4
14
2
8
20
18
22
13
12
6
7
17
5
11
25
32
26
29
24
28
9
34
33
19
16
31
15
10
23
30
21
27
random walk
goodness of split
0.4
0.3
0.2
0.1
0
12
17
7
6
11
5
10
16
15
23
21
19
32
29
28
30
27
24
26
25
22
18
34
33
20
31
9
14
3
13
8
4
2
1
shortest path
without recalculation

Modularity

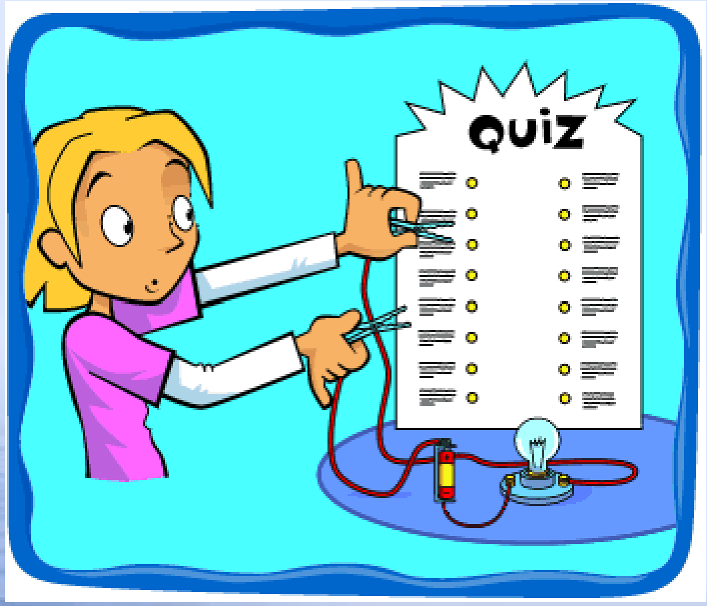
- Spectral clustering essentially tries to *minimize the number edges between groups.*
- *Modularity consider the number edges which is smaller than expected*.

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

- If the difference is significantly large, there's a community structure inside.
- The larger, the better.

Quiz

- *Given a network of m edges, for two nodes with degree k_i, k_j, what is the expected edges between these two nodes?*



Modularity Calculation

$$P_{ij} = \frac{k_i k_j}{2m}.$$

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

- Modularity can be used to determine the number of clusters, why not maximize it directly?
- Unfortunately, it's NP-hard☹

Relaxation

$$\delta(g_i, g_j) = \tfrac{1}{2}(s_i s_j + 1).$$

$$Q = \frac{1}{4m}\sum_{ij}[A_{ij} - P_{ij}](s_i s_j + 1)$$
$$= \frac{1}{4m}\sum_{ij}[A_{ij} - P_{ij}]s_i s_j,$$

$$Q = \frac{1}{4m}\mathbf{s}^T\mathbf{B}\mathbf{s},$$
$$B_{ij} = A_{ij} - P_{ij}.$$

Modularity Matrix

$$\mathbf{s} = \textstyle\sum_{i=1}^{n} a_i \mathbf{u}_i$$

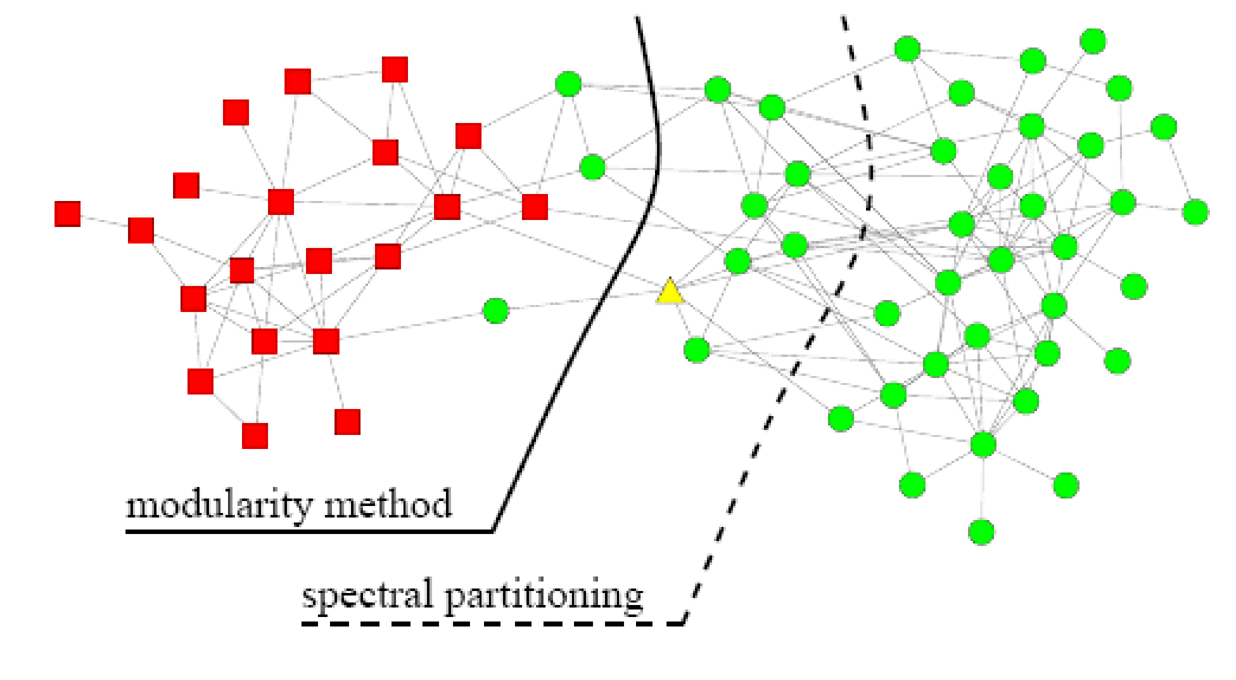
$$Q = \frac{1}{4m}\sum_i a_i^2 \beta_i,$$

Eigen Value Problem!

Beta$_i$ is the eigen value of the Eigen vector u$_i$ of modularity matrix B

Properties of Modularity Matrix

- $$\sum_j B_{ij} = \sum_j A_{ij} - \sum_j P_{ij} = k_i - k_i = 0.$$
- (1,1,…1) is an eigen vector with zero eigen value.
- Different from graph Laplacian, the eigen value of modularity matrix could be +, 0 or -.
- What if the maximum eigen value is zero?
- Essentially, it hints that there’s no strong community pattern. Not necessary to split the network, which is a nice property.



- Here, the spectral partitioning is forced to split the network into approximately equal-size clusters.

Extensions

- Divisive clustering
- K - partitioning…

Comments

- I thought spectral clustering is the end of clustering. But here a new measure Modularity is proposed and found to be working very well, which confirms that "research is endless", or "no last bug".
- Since Graph Laplacian and Modularity matrix both boils down to a eigen value problem, is there any innate connection between these two measures?
- How could it work if we apply it directly to some classic data representation?
- Extend modularity to relational data could be a promising direction.
- There could be more opportunities than "wheels" in social computing.
- Scalability is really a big issue.

References

- M.E.J.Newman, *Finding community structure in networks using the eigenvectors of matrices,* Phys. Rev. ***, 2006***
- ***M. E. J. Newman*, *M. Girvan*, *Finding and evaluating community structure in networks*, Phys. Rev.** 2004