8. MINING SOCIAL GRAPHS

Analyzing Graphs

Up to now

- Objects have been described by attributes
- Relationships among objects inferred from distance measure on attributes

Use of explicit relationships: Graphs

- In many cases relationships are explicitly given
- Prime example: social network graphs
- Graph structure can be inferred from distance measure (e.g. for documents)

So far we have assumed that relationships among different objects in clustering are implicitly defined, using a distance measure that is based on the objects' attributes. In many applications such relationships are not implicitly given, but explicitly provided, notably in social network graphs. There the relationships among users are given by different friend relationships or interactions such as likes and retweets. The resulting graphs can be weighted or unweighted. In the following we will consider the case of unweighted graphs.

Of course, it would also be possible to derive a relationship graph by using a distance measure based on attribute values and using the distance as as an edge weight, respectively consider only edges above a distance threshold. In this way the following methods of clustering objects based on graphs could also be applied in this more traditional context.

Graphs and Clustering

Networks often contain structure

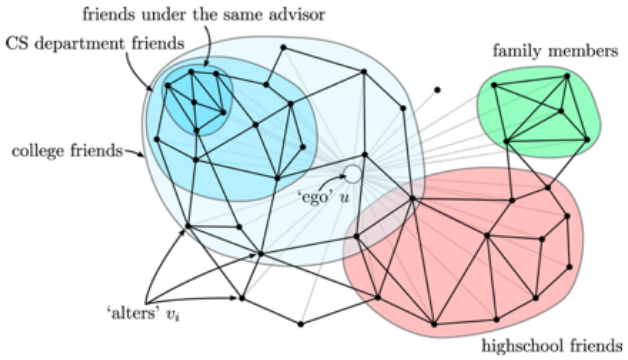
– Clusters (also called communities, modules)

It is widely observed that natural networks contain structure. This is true for social networks (e.g. on social media platforms, citation networks), as well as for many natural networks as we find them in biology. Graph-based clustering aims at uncovering such hidden structures.

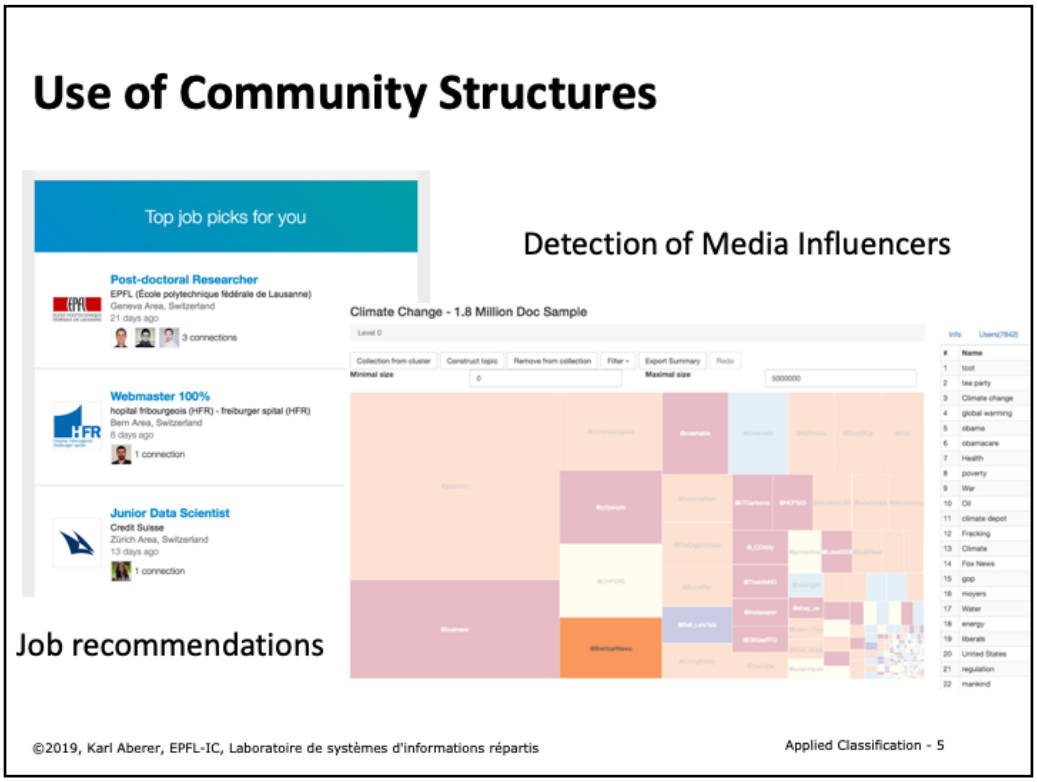
Social Network Analysis

Clusters (communities) in social networks (Twitter, Facebook) related

- Interests
- Level of trust



In social networks mining community structures is particularly popular. Consider the social network neighborhood of a particular social network user, i.e. all other social network accounts that are connected to the user, either through explicit relationships, such as a follower graph, or through interactions such as likes or retweets. Typically the social neighborhood would decompose into different groups, depending on interests and social contexts. For example, the friends from high school would have a much higher propensity to connect to each other, than with members of the family of the user. Thus they are likely to form a cluster. Similarly, other groups with shared interest or high mutual level of trust would form communities.



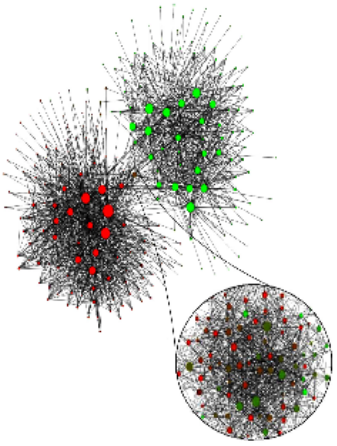
Many tasks can benefit from a reliable community detection algorithm. Online Social Networks rely on their underlying graph to recommend content, for example relevant jobs. Knowing to which community a user belongs can improve dramatically the quality of such recommendations.

Another typical use of community detection is to identify media influencers. Community detection can first be used to detect communities that share in social networks common interests or beliefs (e.g. for climate change we might easily distinguish communities that are climate deniers and climate change believers), and then the main influencers of such communities could be identified.

Use of Community Structures: Social Science

Call patterns in Belgium mobile phone network

– Two almost separate communities

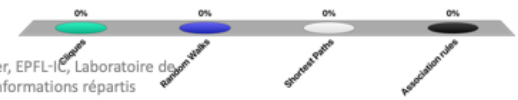


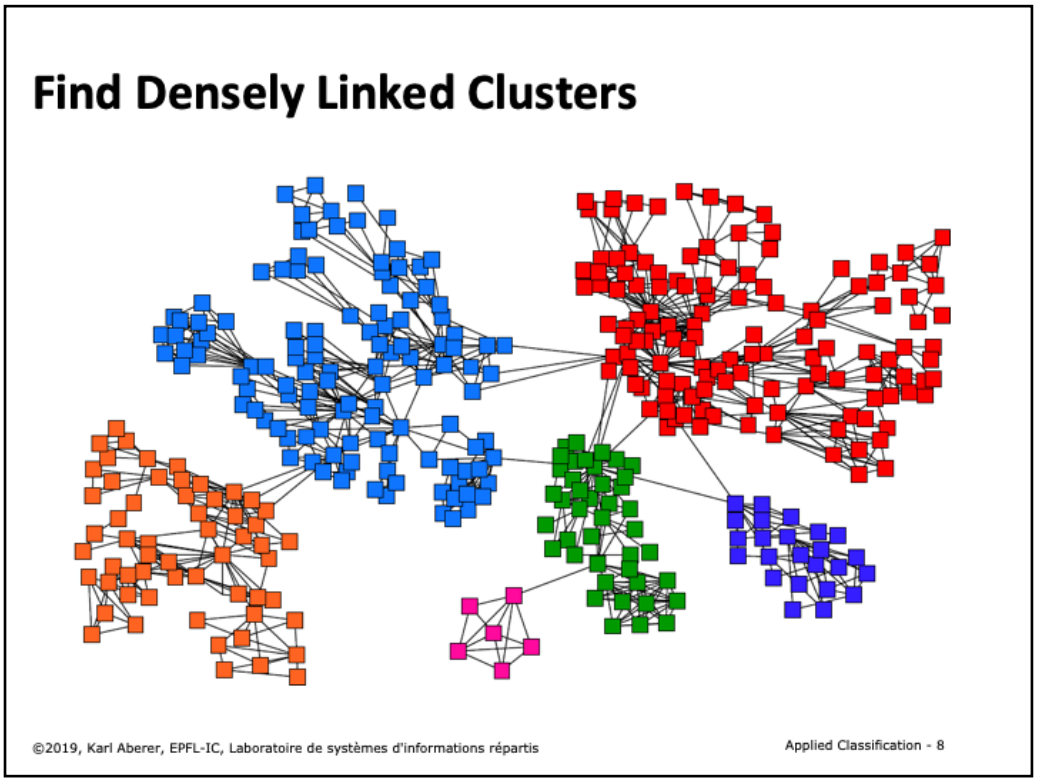
V.D. Blondel et al, *J. Stat. Mech.* P10008 (2008).

In 2008 Vincent Blondel and his students have started applying a new community detection algorithm to the call patterns of one of the largest mobile phone operators in Belgium. It was designed to identify groups of individuals who regularly talk with *each other* on the phone, breaking the whole country into numerous small and not so small communities by placing individuals next to their friends, family members, colleagues, neighbors, everyone whom they regularly called on their mobile phone. The result was somewhat unexpected: it indicated that Belgium is broken into two huge communities, each consisting of many smaller circles of friends. Within each of these two groups the communities had multiple links to each other. Yet, these communities never talked with the communities in the other group (guess why?). Between these two mega-groups was sandwiched in a third, much smaller group of communities, apparently mediating between the two parts of Belgium.

Which of the following techniques do you believe would be most appropriate to identify communities on a social graph?

A. Cliques
B. Random Walks
C. Shortest Paths
D. Association rules





The intuition behind community detection is that the heavily linked components of the graph belong to the same community.

Similarly as earlier for clustering in multidimensional spaces, therefore, the goal of a community detection algorithm is to identify communities that are heavily intra-linked (high intra-cluster similarity) and scarcely inter-linked (low inter-cluster similarity).

Types of Community Detection Algorithms

Hierarchical clustering

- iteratively identifies groups of nodes with high similarity

Two strategies

- **Agglomerative algorithms** merge nodes and communities with high similarity
- **Divisive algorithms** split communities by removing links that connect nodes with low similarity

In general, community detection algorithms are based on a hierarchical approach. The idea is that within communities sub-communities can be identified, till the network decomposes into individual nodes. In order to produce such a hierarchical clustering, two approaches are possible: either by starting from individual nodes, by merging them into communities, and recursively merge communities into larger communities till no new communities can be formed (agglomerative algorithms), or by decomposing the network into communities, and recursively decompose communities till only individual nodes are left (divisive algorithms).

In the following we will present one representative of each of the two categories of algorihtms:

1. The Louvain Algorithm, an agglomerative algorithm
2. The Girvan-newman algorithm, a divisive algorithm

Louvain Modularity Algorithm

Agglomorative Community Detection

- Based on a measure for community quality (**Modularity**)
- greedy optimization of modularity

Overall algorithm

- **first** small communities are found by optimizing modularity **locally** on all nodes
- then each small community is **grouped** into one new community node
- **Repeat** till no more new communities are formed

The Louvain algorithm is essentially based on a the use of a measure, modularity, that allows to assess the quality of a community clustering. The algorithm performs greedy optimization of this measure. It is fairly straightforward: initially every node is considered as a community. The communities are traversed, and for each community it is tested whether by joining it to a neighboring community, the modularity of the clustering can be improved. This process is repeated till no new communities form anymore.

Measuring Community Quality

Communities are sets of nodes with many mutual connections, and much less connections to the outside

Modularity measures this quality: the higher the better

$$\sum_{c \in \text{Communities}} (\#\text{edges within c} - \text{expected } \#\text{edges within c})$$

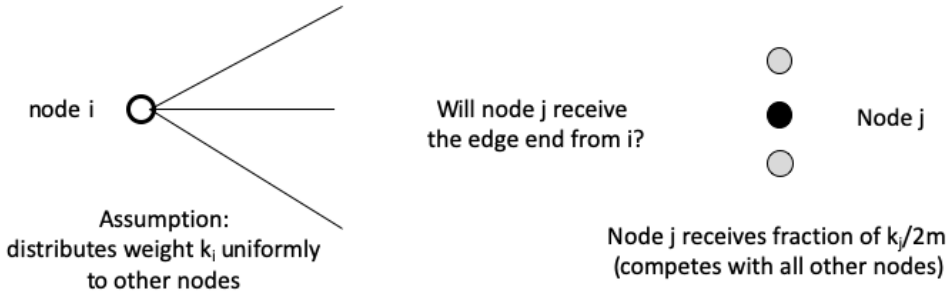
In order to measure the quality of a community clustering, modularity compares simply the difference between the number of edges that occur within a community with the number of edges that would be expected if the edges in the graph would occur randomly. The random graph model is used as what is called the null model, a graph that has the same distribution of node degrees, but randomly assigned connections.

Expected Number of Edges

Graph with unweighted edges

- m = total number of edges
- k_i = number of outgoing edges of node i (degree)

Observation: edges connect to 2m nodes



Assumption:
distributes weight k_i uniformly
to other nodes

Node j receives fraction of $k_j/2m$
(competes with all other nodes)

The null model requires to determining the number of edges that we would expect among a set of nodes, of which we know the degrees, but not the connectivity in detail. Assume that for all nodes i in a community we would now their degree ki (number of edges leaving the node). How many edges would we then observe if the connections on the graph were generated randomly? To answer this question we reason as follows: select one of the nodes i with degree ki. What is now the probability (or fraction of weight) an arbitrary other node j with weight kj would receive? If there are a total of m edges in the network, there are 2m edge ends (since each edge ends in two nodes). If the edge ends of ki are uniformly distributed, node j would thus receive ki/2m edge ends. Thus the number of expected edges connecting nodes i and j would be ki times kj/2m.

Modularity

We define now the modularity measure Q

- A_{ij} = effective number of edges between nodes i and j
- c_i, c_j = communities of nodes i and j

Effective number of edges Expected number of edges

$$Q = \frac{1}{2m}\sum_{i,j}\left(A_{ij} - \frac{k_i k_j}{2m}\right)\delta(c_i, c_j)$$

Properties

- Q in [-1,1]
- 0.3-0.7 < Q means significant community structure

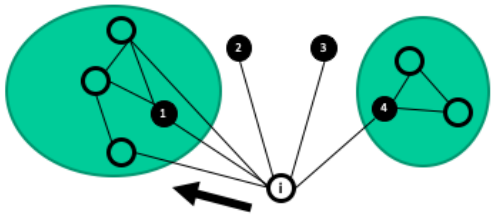
Given the expected number of edges (the null model) we can now formulate the modularity measure as the total difference of number of expected and effective edges for all pairs of nodes from the same cluster. The delta function assures that only nodes belonging to the same community are considered, since it returns 1 when ci = cj.

Due to normalization the measure returns calues between -1 and 1. In general, if modularity exceeds a certain threshold (0.3 to 0.7) the clustering of the network is considered to exhibit a good community structure.

Locally Optimizing Communities

What is the modularity gain by moving node i to the communities of any of its neighbors?

– Test all possibilities and choose the best

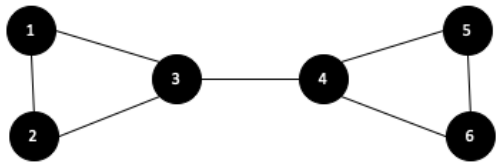


Given the modularity measure the next question is now how to use it to infer the communities. This is performed through local optimization. The algorithm sequentially traverses all nodes of the network, and for each node checks how the modularity can be increased maximally by having the node joining the node of a neighboring community. It then decides to join that node to the best community.

Example

Initial modularity Q = 0

Start processing nodes in order



We illustrate the algorithm for a simple example. Initially, the modularity is zero since all nodes belong to different communities. We start now to process the nodes in some given order, e.g. size of identifier.

Example: Processing Node 1

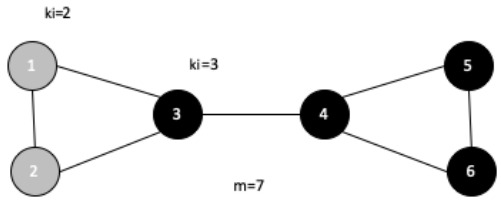
Joining node 1 to node 2

- Q = 1/2*Ⓜ(1 –[2]*[2]/2*Ⓜ) = 1/14 (1 - 4/14) = 1/14 * 10 /14 > 0

Joining node 1 to node 3

- Q = 1/14 (1 –[2]*[3]/2*7) = 1/14 (1 – 6/14) = 1/14 * 8/14 > 0

New modularity: 1/14 * 10 /14



For node 1 we can join it either to node 2 or 3, it's two neighbors. To decide which is better, we have to compute modularity after the join. We see that it is better to join node 2, as the resulting modularity will be higher. We can think of this as follows: since node 3 has more connections, it is more likely to be randomly connected to node 1, this connecting to node 2 is more "surprising".

Example: Processing Nodes 2 and 3

Joining node 2 to node 3 (leaving community of Node 1)

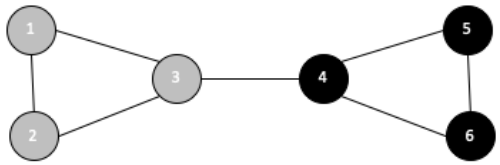
- no improvement

Joining node 3 to community {1,2} (via node 1 or 2)

- Q = 1/14 (3 – 4/14 – 6/14 – 6/14) = 1/14 * 26 / 14

Joining node 3 to node 4

- Q = 1/14 10/14 + 1/14 (1 – 9/14) = 1/14 * 15/14



For node 2 there will be no change, as the only alternative would be node 3. But for the same reasons node 1 did not join node 3, also node 2 does not. Next is node 3: here we have two choices, either to join community {1,2}, or to join node 4. In the first case we obtain a larger community, and in the second case two smaller communities. Computation of modularity reveals that joining 1 and 2 gives a much better community structure.

Example: Processing Nodes 4, 5 and 6

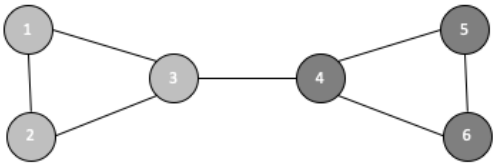
Joining node 4 to community {1,2,3} via 3

- Q = 1/14 (4 - 4/14 - 6/14 - 6/14 – 6/14 – 6/14 - 9/14) = 1/14 * 19/14

Joining node 4 to node 5

- Q = 1/14 * 26/14 + 1/14 * (1 – 6/14) = 1/14 * 34/14

Finally, also node 6 will join the second community



Similar arguments as before will then join node 4 to 5 and finally node 6 to the community consisting of nodes 4 and 5. Then the first round of processing is over.

Example: Merging Nodes

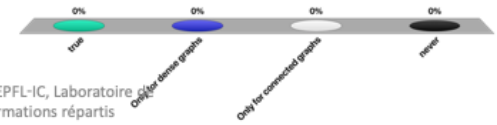
Now that all nodes have been processed, we merge nodes of the same community in a single new nodes and restart processing

Will the two remaining nodes merge? Answer: yes

For the next round of processing the resulting community nodes are collapsed in new nodes for the complete community, and the algorithm is re-run with the new resulting graph. Obviously, now the two remaining nodes will merge into a community, as the modularity will move from zero to positive. Then the algorithm terminates.

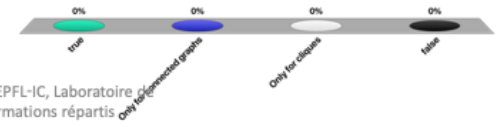
Modularity clustering will end up always with a single community at the top level?

A. true
B. Only for dense graphs
C. Only for connected graphs
D. never



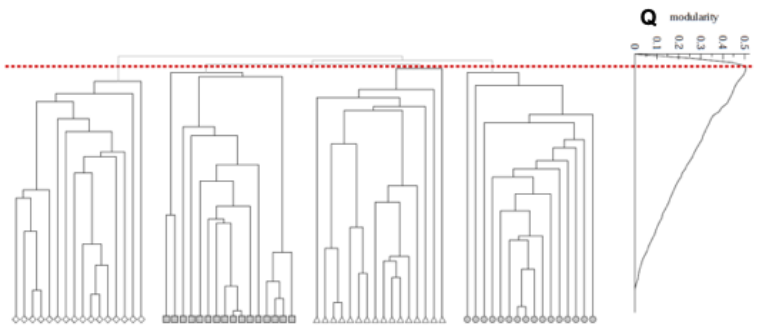
Modularity clustering will end up always with the same community structure?

A. true
B. Only for connected graphs
C. Only for cliques
D. false



Modularity to Evaluate Community Quality

Modularity can also be used to evaluate the best level to cutoff of a hierarchical clustering



Apart from constructing the communities, the modularity measure can also be used to evaluate the quality of communities in hierarchical clustering. This can be done independently of how the clustering has been constructed. In fact, there is an optimal level of clustering when moving from one level of the hierarchy to the next. Initially increasing the number of communities increases their quality, by separating distinct communities. At a certain point, splitting of communities in smaller communities worsens the quality of the community structure. Thus there exists an optimum point of clustering that can be selected using modularity.

Louvain Modularity - Discussion

Widely used in social network analysis and beyond

- Method to extract communities from very large networks very fast

Complexity: O(n log n)

Louvain modularity clustering is today the method of choice for social network clustering, mainly because of its good computational efficiency. It runs in n log n, which makes it applicable for very large networks, as they occur today, in particular for social networks resulting from large platforms, as social network sites, messaging services or telephony.

Girvan-Newman Algorithm

Divisive Community Detection

- Based on a **betweenness measure** for edges, measuring how well they separate communities
- Decomposition of network by splitting along edges with highest separation capacity

Overall algorithm

- Repeat until no edges are left
  - Calculate betweenness of edges
  - Remove edges with highest betweenness
  - Resulting connected components are communities
- Results in hierarchical decomposition of network

We now introduce a second algorithm fro community detection, that belongs to the class of divisive algorithms. Also this algorithm is based on a measure, this time on a measure on edges. The betweenness measure gives an indication which edges are likely to connect different communities, and thus are good splitting points, to partition larger parts of the network in to communities. The algorithm recursively decomposes the network, by removing edges with the highest betweenness measure, till no edges are left. Also this algorithm results in a hierarchical clustering.

Edge Betweenness

Edge betweenness: fraction of number of shortest paths passing over the edge

$$betweenness(v) = \sum_{x,y} \frac{\sigma_{xy}(v)}{\sigma_{xy}}$$

where

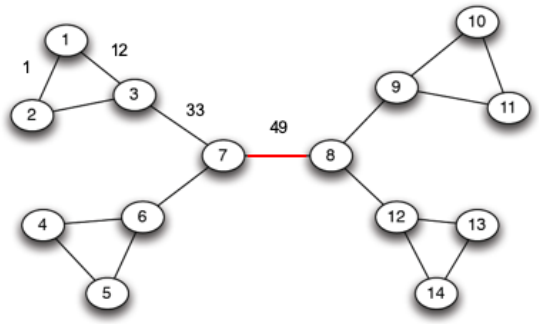
σ_{xy}: number of shortest paths from x to y

$\sigma_{xy}(v)$: number of shortest paths from x to y passing through v

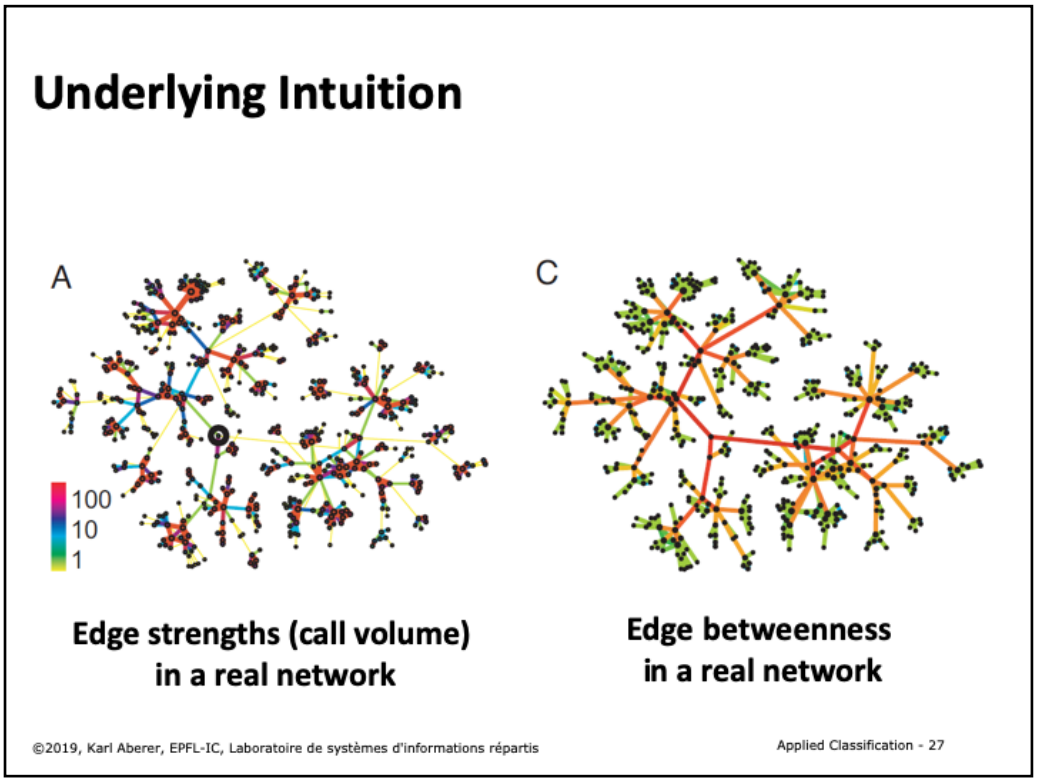
Betweenness centrality is an indicator of a node's centrality in a network. It is equal to the number of shortest paths from all vertices to all others that pass through that node. A node with high betweenness centrality has a large influence on the transfer of items through the network, under the assumption that item transfer follows the shortest paths. The concept finds wide application, including computer and social networks, biology, transport and scientific cooperation.

Alternatively, there exist also the concept of *random-walk betweenness*. A pair of nodes m and n are chosen at random. A walker starts at m, following each adjacent link with equal probability until it reaches n. Random walk betweenness x_{ij} is the probability that the link $i \rightarrow j$ was crossed by the walker after averaging over all possible choices for the starting nodes m and n

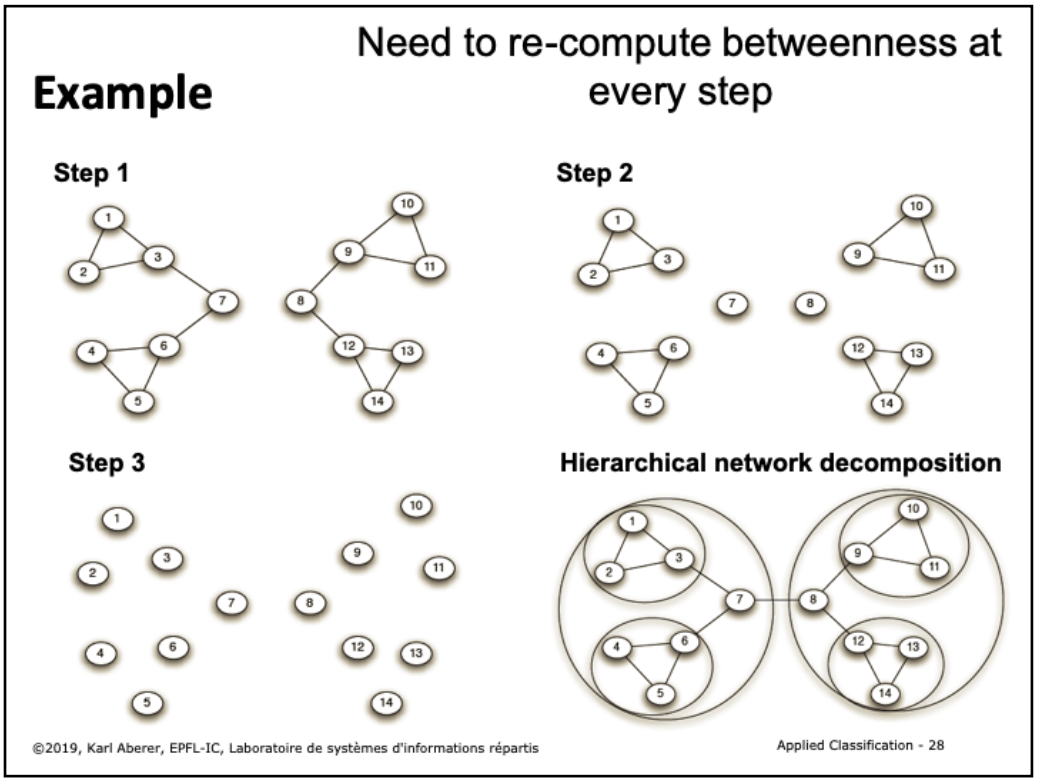
Example



In this example network we compute betweenness for some selected edges. For example, for the edge 1-2 there is one shortest path between 1 and 2 that traverses the edge 1-2, thus the value is 1. For 1-3 there are shortest paths from the remaining 12 nodes in the network (except node 2) to node 1 that have to pass through this edge, thus the betweenness value is 12. For edge 3-4 we have paths going to both nodes 1 and 2, thus the betweenness value of that edge is significantly higher than for 1-3.



In a sense betweenness is the dual concept to connectivity. This is illustrated by the two graphs that result from real phone call networks. On the left hand side we see the indication of the strengths of connections among the nodes. Communities are tightly connected by such links. On the right hand side we see the betweenness measure. Now the links that are connecting communities have a high strength, since, intuitively speaking, the traffic from one community to another has to traverse over these sparsely available links.



Here we illustrate the execution of the Girvan-Newman algorithm. In Step 1 we remove one edge (in the middle) that had the highest betweenness value, resulting in two communities. Next (by symmetry) the edges connected to nodes 7 and 8 are removed, and in the third and fourth step the network decomposes completely. By overlaying the communities that have resulted from each step we obtain the final hierarchical clustering.

As the graph structure changes in every step, the betweenness values have to be recomputed in every step. This constitutes the main cost of the algorithm.

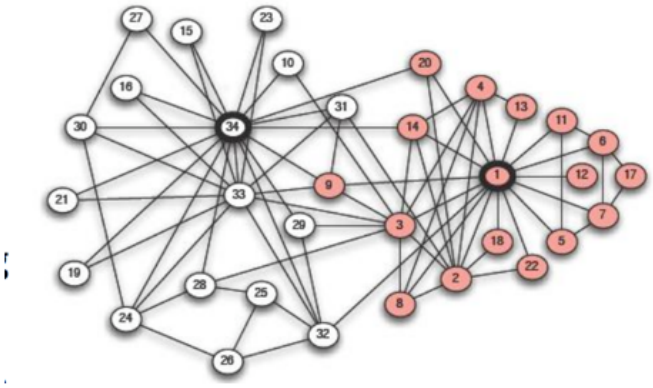
Girvan-Newman: Sample Results

Communities in physics collaborations

The algorithm has been applied in many contexts, in particular on smaller graphs resulting from social science studies.

Girvan-Newman: Sample Results

Zachary's Karate club: Hierarchical decomposition

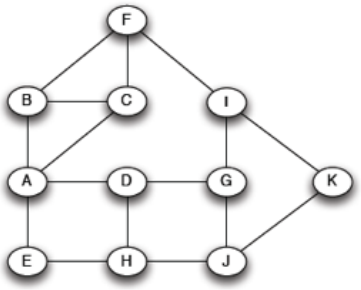


The origins of the algorithm come from studying a network of the 34 karate club members studied by the sociologist Wayne Zachary. Links capture interactions between the club members outside the club. The circles and the squares denote the two fractions that emerged after the club split into two following a feud between the group's president and the coach. The split between the members closely follows the boundaries of these communities. This karate club has been historically used as a benchmark to test community finding algorithms.

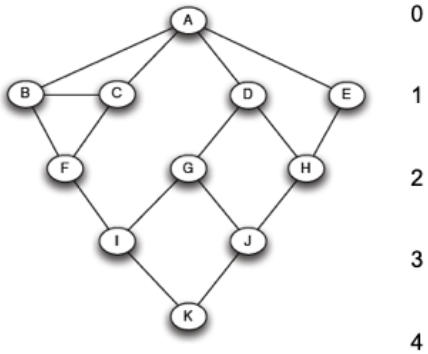
There was one outlier, node number 3, where the algorithm wrongly assigns the member. at the time of conflict, node 9 was completing a four-year quest to obtain a black belt, which he could only do with the instructor (node 34)

Computing Betweenness - BFS

Computing betweenness of paths starting at node A



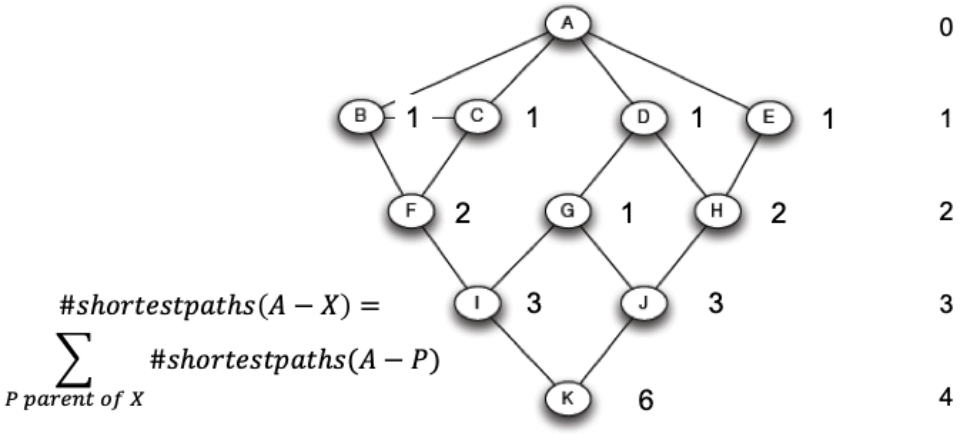
Perform BFS starting from A



We describe now the process of computing the betweenness values. The approach is to perform a breadth-first search (BFS) for every node in the graph. The nodes are arranged in increasing levels of distance of the starting node, e.g. node A.

Computing Betweenness – Path Counting

Count the number of shortest paths from A to all other nodes of the network



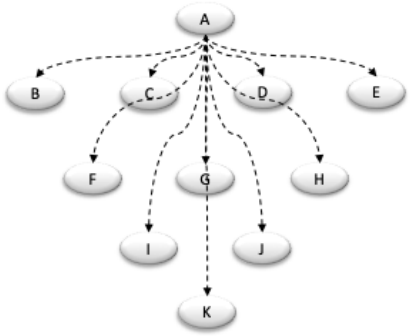
$$\#shortestpaths(A-X) = \sum_{P\ parent\ of\ X} \#shortestpaths(A-P)$$

In a first phase we count the number of shortest paths that are leading to each node, starting from node A. To do so we can simply reuse the data that has been computed at the previous level, summing up the number of paths that have been leading to each parent of a given node.

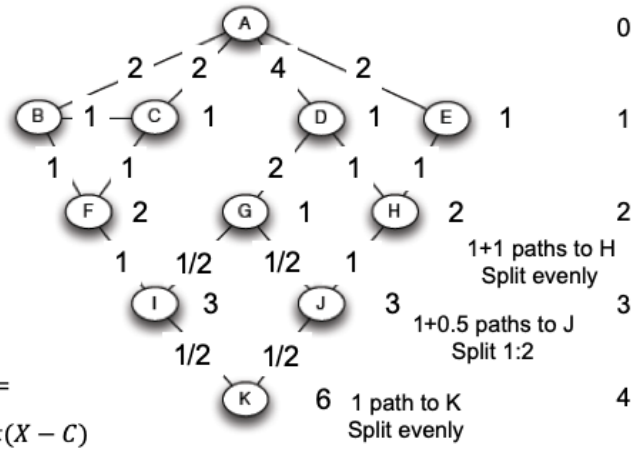
Computing Betweenness – Edge Flow

Edge Flow

- 1 unit of flow from A to each node
- Flow to be distributed evenly over all paths
- Sum of the flows from all nodes equals the betweenness value



Computing Edge Flow



$$weighttosplit(X) = 1 + \sum_{C\ child\ of\ X} edgeweight(X - C)$$

In a second phase we compute edge flow values (that are the betweenness values related to node A). We proceed in a bottom-up approach. The flow that arrives at every node is 1. In addition, the node receives also all the flows that are passing on to the children. Thus, the total flow weight a node receives (or has to pass through) is 1 plus the flows to all of its children. This weight is distributed over the parents, proportionally to the number of paths that are leading to those parents (what has been computed in phase 1).

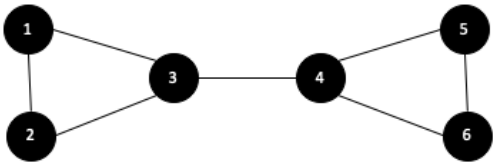
Algorithm for Computing Betweenness

1. Build one BFS structure for each node
2. Determine edge flow values for each edge using the previous procedure
3. Sum up the flow values of each edge in all BFS structures to obtain betweenness value
   - Flows are computed between each pairs of nodes → final values divided by 2

Once the flows specific to a node have been computed for every node, the last step is to aggregate for each edge all the flow values that have been computed for all the nodes. In this way we compute each flow twice (flow into both direction), therefore, the final betweenness value corresponds to the aggregate flow value divided by 2.

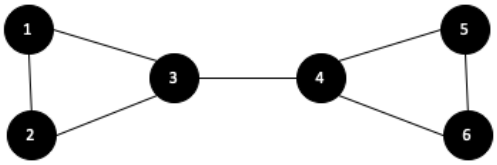
Betweeness of edge 3-4 is …

A. 16
B. 12
C. 9
D. 4



When computing path counts for node 1 with BFS, the count at 6 is …

A. 1
B. 2
C. 3
D. 4



Girvan-Newman Discussion

Classical method

- Works for smaller networks

Complexity

- Computation of betweenness for one link: $O(n^2)$
- Computation of betweenness for all links: $O(L\ n^2)$
- Sparse matrix: $O(n^3)$

The Girvan-Newman algorithm is the classical algorithm for community detection. It's major drawback is its scalability. The flow computation for one link has quadratic cost in the number of nodes (it has to be computed for each pair of nodes). If we assume sparse networks, where the number of links is of the same order as the number of nodes, the total cost is cubic. This was also one of the motivations the inspired the development of the modularity based community detection algorithm

References

The slides are loosely based also on:

- http://barabasilab.neu.edu/courses/phys5116/

Papers

- Blondel, Vincent D., et al. "Fast unfolding of communities in large networks." *Journal of statistical mechanics: theory and experiment* 2008.10 (2008): P10008
- Girvan, Michelle, and Mark EJ Newman. "Community structure in social and biological networks." *Proceedings of the national academy of sciences* 99.12 (2002): 7821-7826.

References

Document classification

- Aggarwal, Charu C., and Cheng Xiang Zhai. "A survey of text classification algorithms." *Mining text data*. Springer US, 2012. 163-222.
- Jindal, Rajni, Ruchika Malhotra, and Abha Jain. "Techniques for text classification: Literature review and current trends." *Webology* 12.2 (2015): 1
- Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press. 2008 (http://www-nlp.stanford.edu/IR-book/) Chapter 13
- Fasttext: https://www.youtube.com/watch?v=CHcExDsDeHU

Recommender Systems

- Chapter 9 in mmds.org
- Recommender Systems Handbook, Springer 2015