3.3 GRAPH MINING

Mining Graphs

Data Mining can be performed on different types of data

- Structured data: tables, graphs
- Unstructured data: text, images, sensor data

Graph data is increasingly important

- Social networks and Web
- Knowledge Graphs
- Scientific data on networks
- Graph structure can also be inferred from distance measures (e.g., for documents)

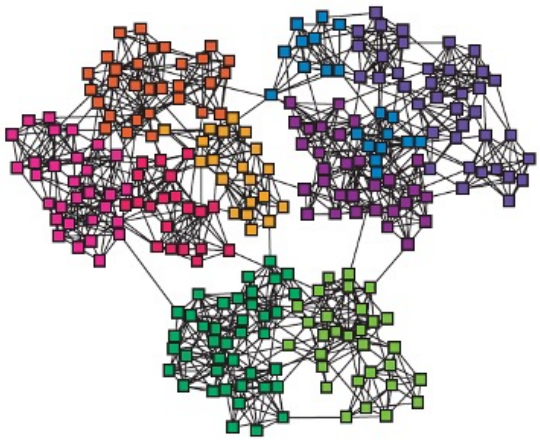
Data mining is applied for a wide range of both structured and unstructured data. It has traditionally evolved from analyzing large transactional databases, typically for business applications,. Structured data graph data is playing an increasingly important role in data mining, due to a growing number of graph data sources. One area where graph data play an obvious role is in social networks, where social interactions and relationships are modelled as graphs. This availability of graph data and the need to understand the structure of large graphs have been driving factors in the development and wide-spread application of graph mining algorithms.

It is also worthwhile to note that graph mining algorithms can be applied to graphs that are generated from other data types. For example, document similarity measures based on text embeddings could be used to generate graph structures for document collections.

Graphs and Clustering

Graphs often contain structure

– Clusters (also called communities, modules)

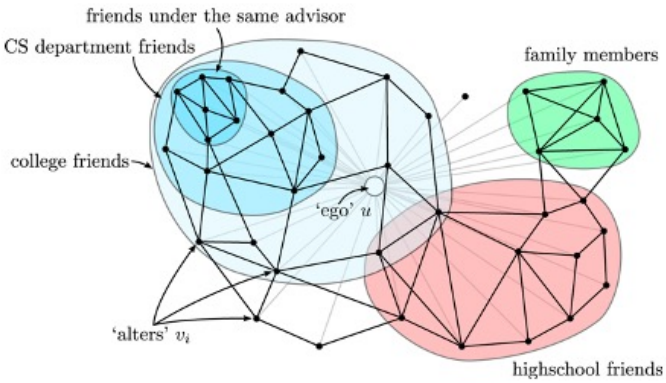


It is widely observed that natural networks contain structure. This is true for social networks (e.g., social media platforms, citation networks), as well as for many natural networks as we find them in biology. Graph-based clustering aims at uncovering such hidden structures.

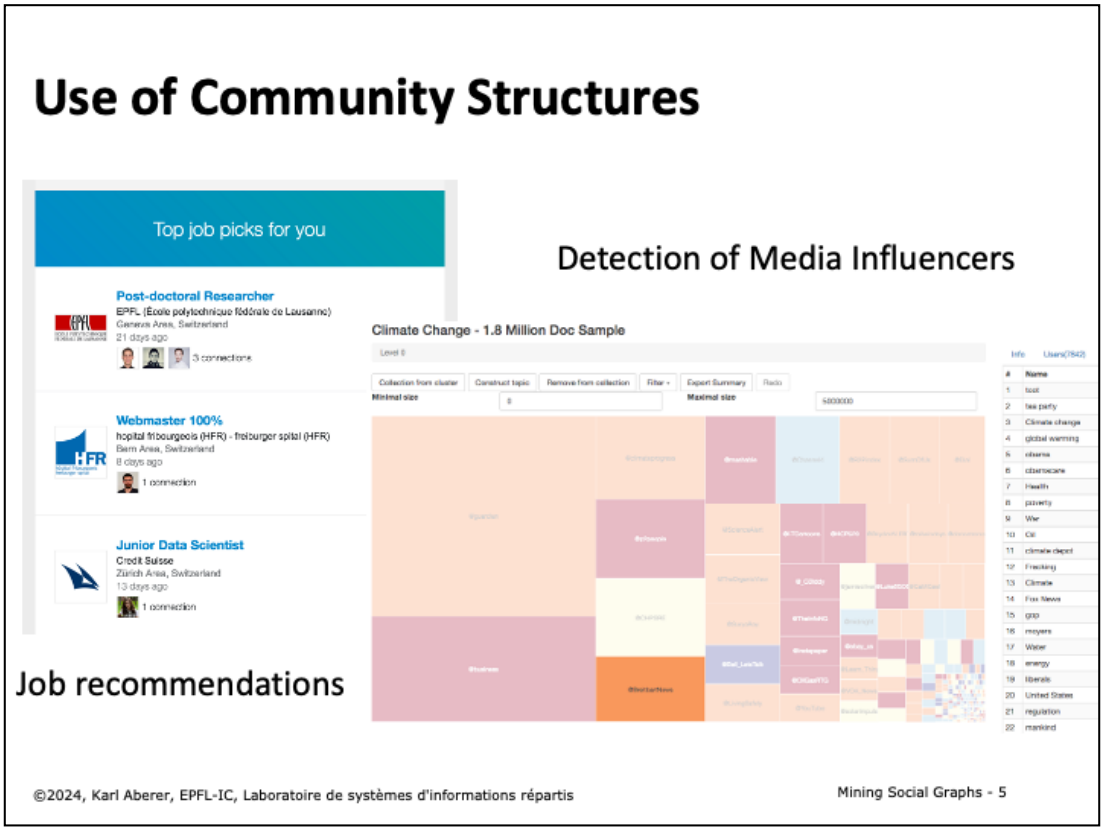
Social Network Analysis

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

- 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. Graph mining is a tool to detect these types of communities.



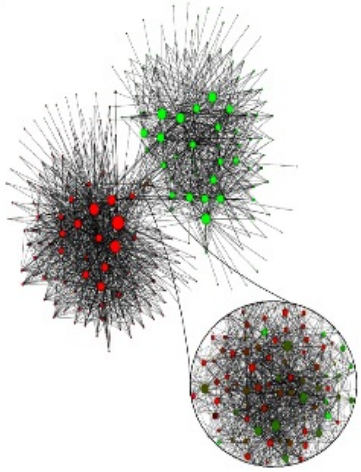
Many tasks can benefit from a reliable community detection algorithm. Online Social Networks rely on their underlying graph to recommend content, for example relevant jobs. Knowing to which community a user belongs can improve dramatically the quality of such recommendations.

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., in the discussion on 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 cell phone network

– Two almost separate communities



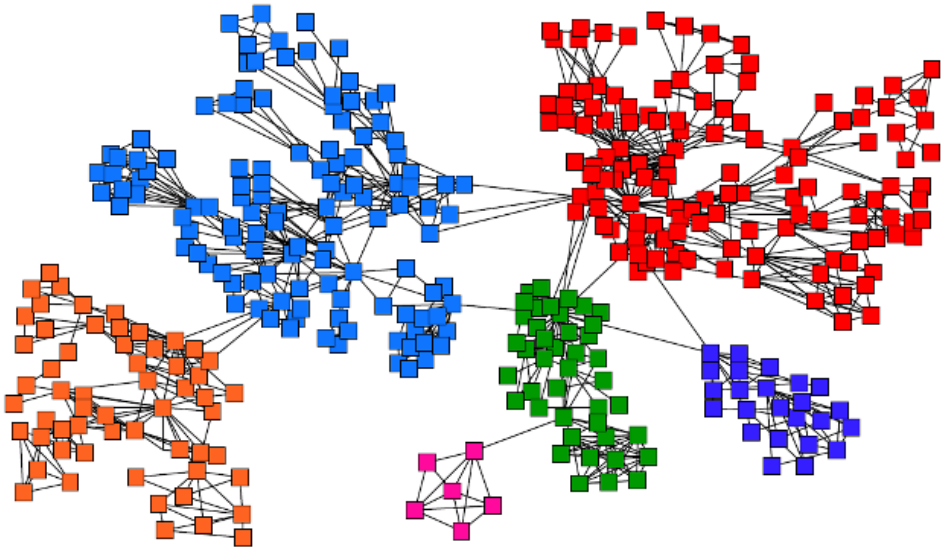
V.D. Blondel et al, *J. Stat. Mech.* P10008 (2008).

In 2008 Vincent Blondel and his students have started applying a new community detection algorithm to the call patterns of one of the largest cell 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, whom they regularly called on their cell 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 large groups we find a third, much smaller group of individuals, apparently mediating between the two parts of Belgium.

Which of the following graph analysis 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

Task: Find Densely Linked Clusters



The intuition behind community detection is that the heavily linked components of the graph belong to the same community. Thus, a community is a set of nodes that has many connections among themselves, but few to other parts of the network. These communities form clusters in the network.

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 taking a hierarchical approach. The idea is that in a community smaller 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 algorithms:

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

3.3.1 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 the use of a measure, modularity, that allows to assess the quality of a community clustering. The algorithm performs greedy optimization of this measure. The basic idea is 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.

A short remark on terminology: in mathematics the usual terminology for graph nodes is “vertex”. We will in the following continue to use the term “node” for “vertex”, as it is custom in computer science.

Measuring Community Quality

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

Modularity measures this quality: the higher the better

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

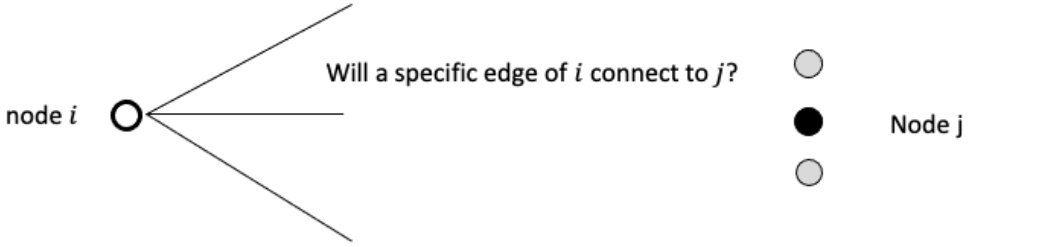
In order to measure the quality of a community clustering, modularity compares the difference between the number of edges that occur within a community with the number of edges that would be expected in a random subset of nodes of the same size and randomly distributed edges.

Expected Number of Edges

Graph with unweighted edges

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

Observation: there exist $2m$ "edge ends"



distributes k_i edge ends uniformly to other nodes

$\frac{k_j}{2m-1}$ edges end up in node j

Expected number edges of node i ending up in node j: $k_i \frac{k_j}{2m-1} \approx \frac{k_i k_j}{2m}$

We now determine the number of edges we expect between nodes with given degrees ki, if edges were purely randomly allocated.

How many edges would we 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 the probability that this edge connects to another node j with weight kj? 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 are uniformly distributed and there are 2m-1 remaining edge ends in the graph, the probability to connect exactly to node j would be kj/2m-1. Applying the same argument to all outgoing edges of node i, node j will connect to node i with probability ki*(kj/2m-1). For large m we can replace 2m-1 by 2m.

Modularity

Modularity measure Q

- A_{ij} = effective number of edges between nodes i and j
- C_i, C_j = clusters of nodes i and j

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

Expected number of edges

Effective number of edges

Properties

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

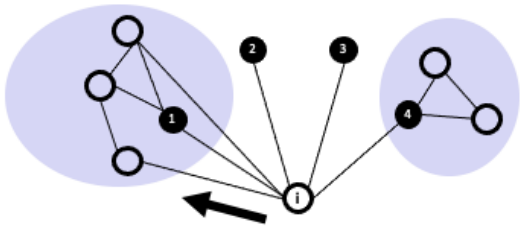
Given the expected number of edges we can now formulate the modularity measure as the total difference of number of expected and effective edges for all pairs of nodes from the same cluster. The delta function assures that only nodes belonging to the same cluster are considered, since it returns 1 when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

Due to normalization the measure returns values 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 one that increases modularity the most, if it exists

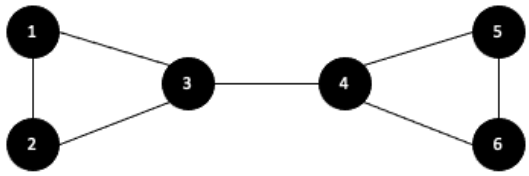


Given the modularity measure, the next question is now how to use it to infer the communities. This is performed through local optimization. The algorithm sequentially traverses all nodes of the network, and for each node checks how the modularity can be increased maximally by having the node joining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

Example: Processing Node 1

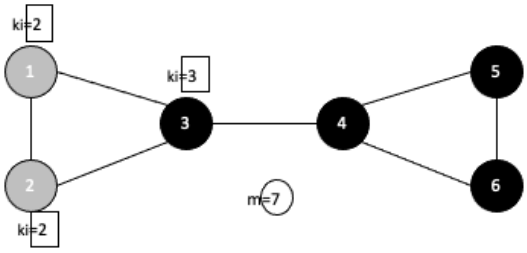
Joining node 1 to node 2

- Q = 1/2*7 (1 – 2*2/2*7) = 1/14 * 10 /14 > 0

Joining node 1 to node 3

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

New modularity: 1/14 * 10 /14



Node 1 can either join node 2 or 3, it's two neighbors. To decide which is better, we compute modularity after the join. We see that joining node 2 results in a higher modularity. We can interpret this as follows: since node 3 has more connections, it is more likely to be randomly connected to node 1, thus connecting to node 2 is less likely to be the result of a random connection.

Example: Processing Nodes 2 and 3

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

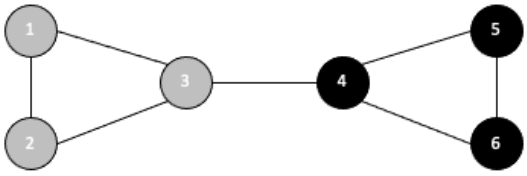
- no improvement

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



Node 2 will not change the community, 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. The next node is node 3: this node has two choices, either to join community {1,2}, or to join node 4. In the first case we obtain one larger community, and in the second case two smaller communities. Computation of modularity reveals that joining join community {1,2}, gives a higher modularity.

Example: Processing Nodes 4, 5 and 6

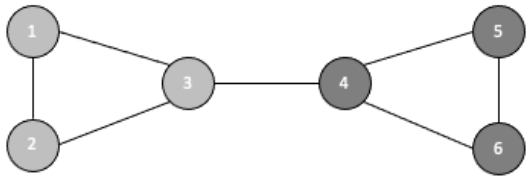
Joining node 4 to community {1,2,3} via 3

- 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



For similar reasons as before, node 4 will join node 5 and finally node 6 will join nodes {4,5}. This completes the first iteration.

Example: Merging Nodes

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

Will the two remaining nodes merge? Answer: yes

In the next iteration, the current community nodes are collapsed into new nodes representing the communities, and the algorithm is re-run with the new resulting graph. Obviously, now the two remaining nodes will merge into a single community, as the modularity will change from zero to a positive value. Then the algorithm terminates.

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

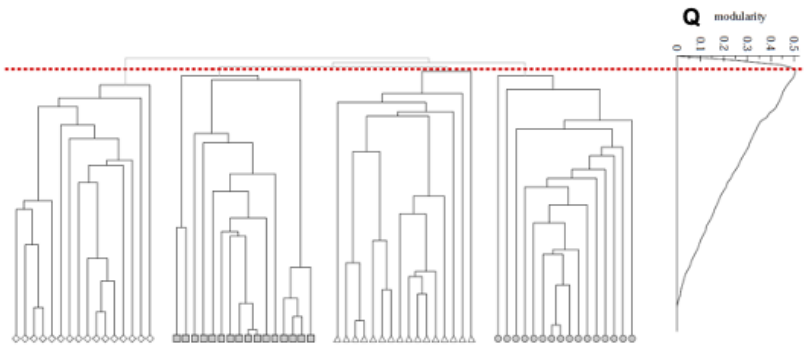
A. true
B. Only for dense graphs
C. Only for connected graphs
D. never

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 detecting communities, the modularity measure can also be used to evaluate the quality of communities in a hierarchical clustering. This can be done independently of how the clustering has been constructed.

By evaluating modularity at each level of the hierarchy, an optimal modularity value can be determined. This optimal value indicates which are the highest quality communities, and lower levels in the tree hierarchy can be pruned.

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 widely used method for social network clustering, mainly because of its good computational efficiency. It runs in O(n log n), which makes it applicable for very large networks, for example, for social networks resulting from large Internet platforms, such as social networking sites or messaging services.

3.3.2 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 for community detection, that belongs to the class of divisive algorithms. This algorithm uses he betweenness measures that quantifies the capacity of an edge to separate the network into distinct subnetworks. Note that while for an agglomerative algorithm we used a measure that quantifies how well communities are connected, for a divisive algorithm we use a measure that quantifies how well communities can be separated.

Using the betweenness measure, the algorithm will recursively split the network into smaller networks, till arriving at individual nodes. This will result in a hierarchical clustering.

Edge Betweenness Centrality

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

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

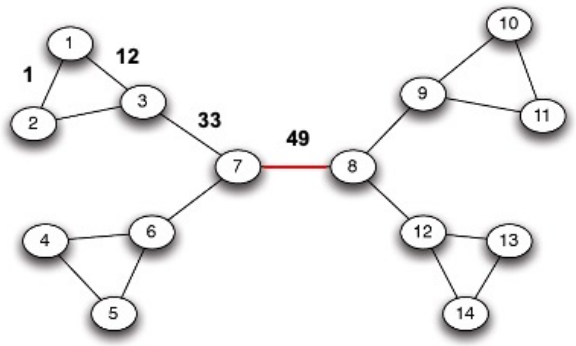
where

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 nodes to all other nodes 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.

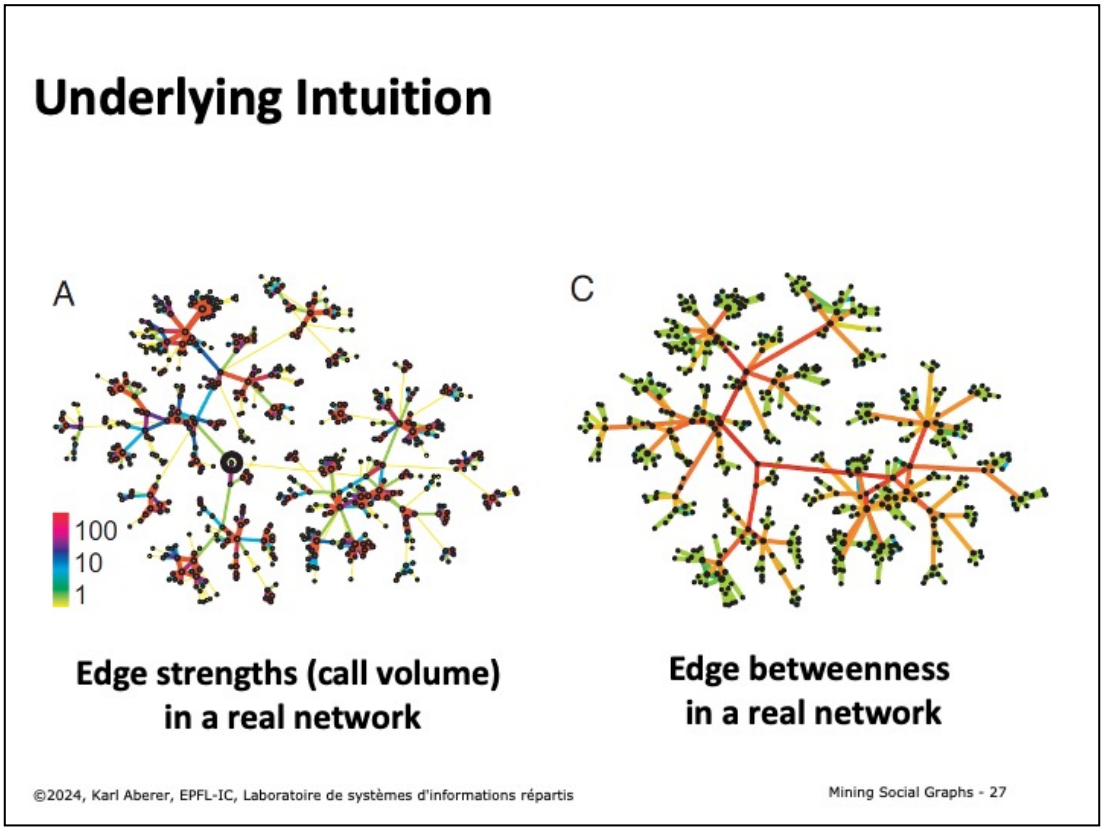
As an alternative measure, one could also consider *random-walk betweenness*. A pair of nodes x and y are chosen at random. A walker starts at x, following each adjacent link with equal probability until it reaches y. Random walk betweenness r_{xy} is the probability that the link $x \rightarrow y$ was crossed by the walker after averaging over all possible choices for the starting nodes x and y. Different to betweenness this measure does consider all paths between the two nodes, not only the shortest paths. However, this measure is expensive to compute.

Example

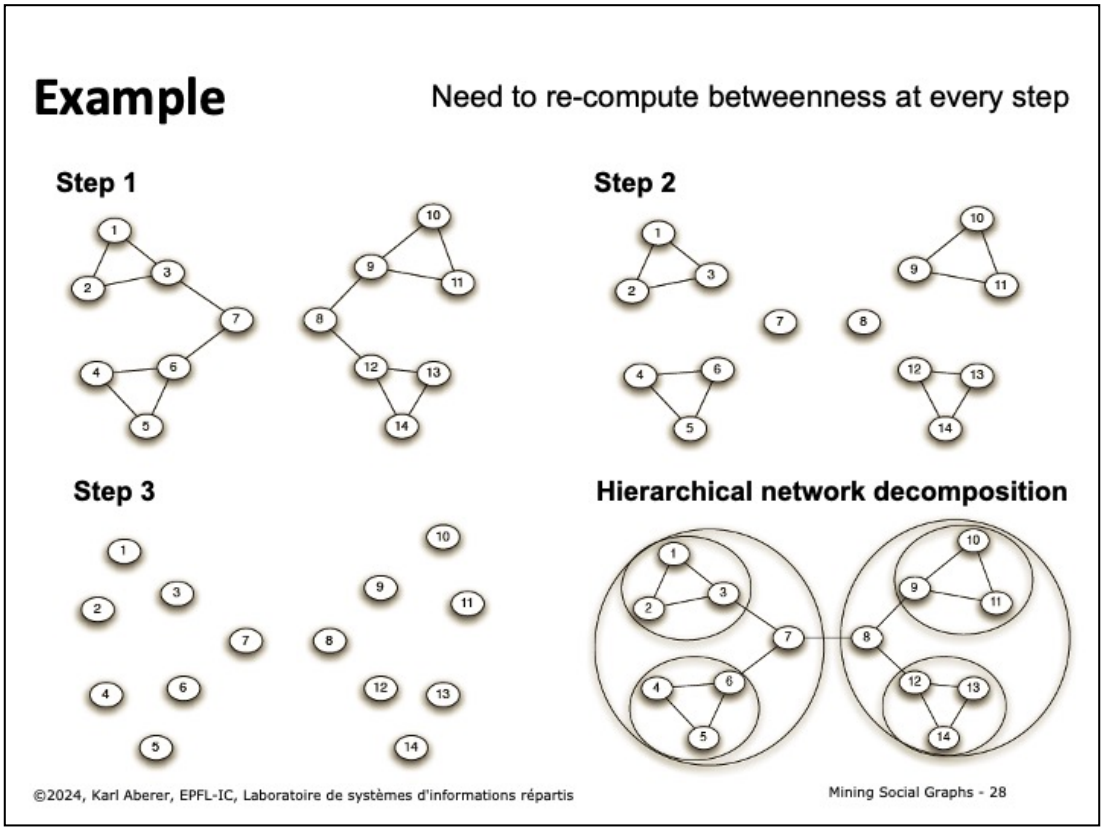


$$\sigma_{xy}(v)$$

In this example network we illustrate the value of $\sigma_{xy}(v)$ 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 must pass through this edge, thus the betweenness value is 12. For edge 3-7 we have shortest paths reaching both nodes 1 and 2, thus the betweenness value of that edge is significantly higher than of 1-3. For edge 7-8 we have 7 nodes in each of the two subnetworks on the left and on the right (including the nodes 7 and 8). Among each pair of nodes there exists exactly one shortest path. Therefore, the value of $\sigma_{xy}(v)$ is 49.



Betweenness can be considered as a concept that is dual 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 different communities have a high strength. This can be interpreted in different ways: on the one hand traffic from one community to another has to traverse these links, on the other hand if such links are cut, communities are separated.

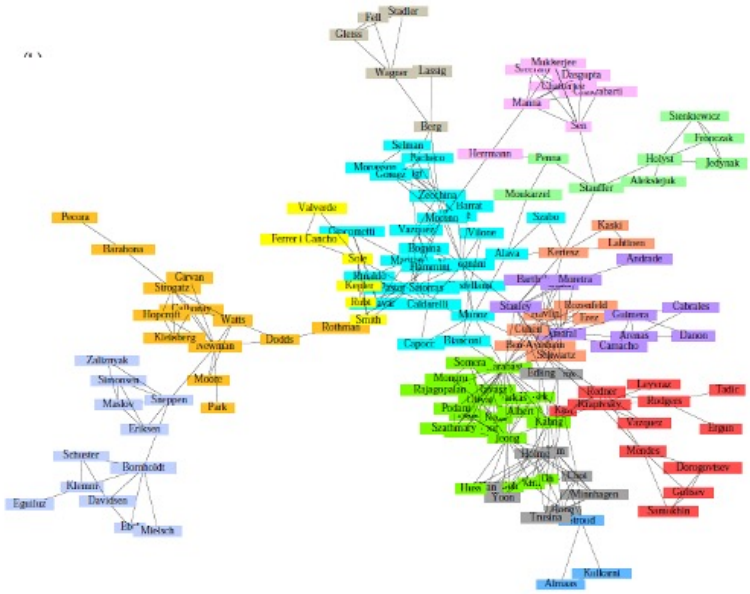


Next, we illustrate the principle of the Girvan-Newman algorithm. It proceeds by recursively removing edges with highest betweenness from the network.

In Step 1 it removes one edge (in the middle) that had the highest betweenness value, resulting in two communities. Next 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 need to be recomputed in every step. This results in the main cost of the algorithm. We will next explore the question of efficient computation of betweenness.

Girvan-Newman: Sample Results

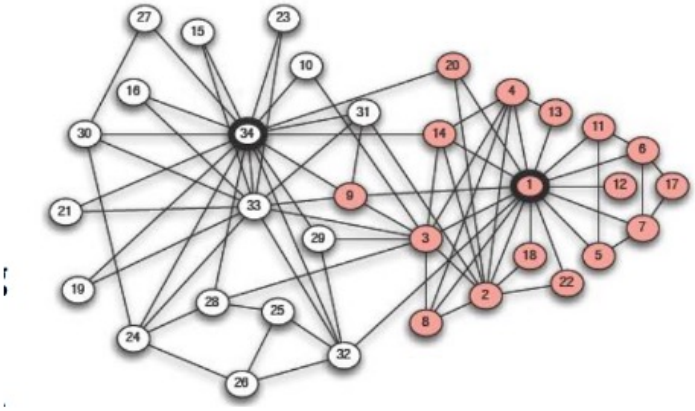


Communities in physics collaborations

The algorithm has been applied in many contexts, due to the computational cost mostly on smaller graphs resulting from social science studies.

Girvan-Newman: Sample Results

Zachary's Karate club: Hierarchical decomposition



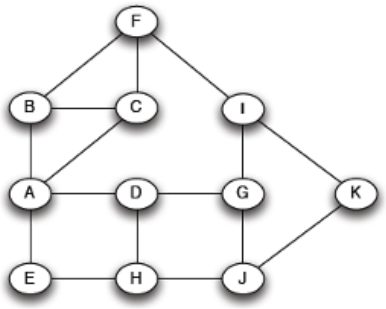
In fact, the origin of the Girvan-Newman algorithm can be traced back to a specific social science study. In this study a network of 34 karate club members was studied by the sociologist Wayne Zachary Links capture interactions between the club members outside the club. The white and gray nodes denote the two communities that resulted after the club decided to split following a feud between the group's president and the coach. The split between the members closely follows the boundaries of the two communities. In the community graph we can identify one outlier, node number 9, 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.

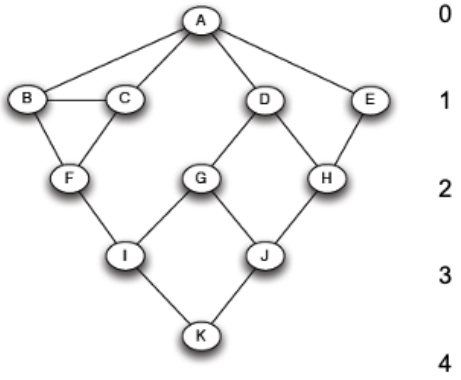
This example has been widely used as a benchmark to test community detection algorithms.

Computing Betweenness - BFS

Computing betweenness of paths starting at node A

Perform BFS starting from A

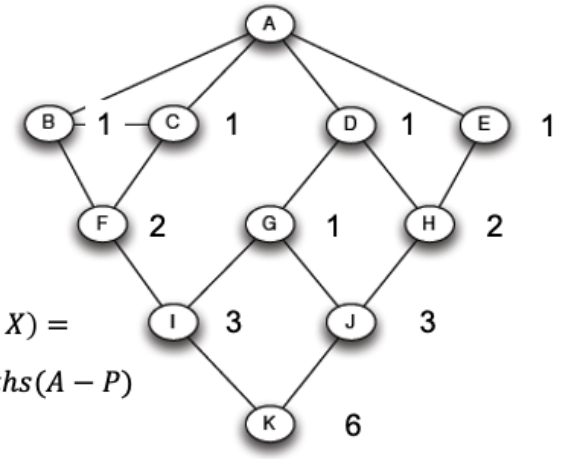




We describe now the approach for computing the betweenness values. It is based on a breadth-first search (BFS) starting at every node in the graph. For a given node, e.g., node A, the other nodes are arranged in levels according to increasing distance from the node.

Computing Betweenness – Path Counting

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



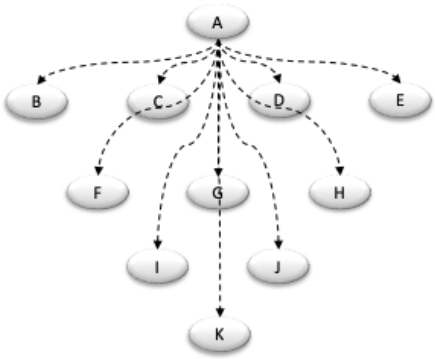
$$\#shortestpaths(A-X) = \sum_{P\ parent\ of\ X} \#shortestpaths(A-P)$$

In a first phase, we count the number of shortest paths that are leading to each node, starting from node A. To do so, the algorithm can at each level reuse the data that has been computed at the previous level. The number of shortest paths leading to a given node corresponds to the sum of number of shortest paths leading to each of its parent needs. For example, the shortest paths leading to node F, are all shortest paths leading to nodes B and C. Note that in this way paths that are not shortest paths are ignored, like the path A-B-C-F.

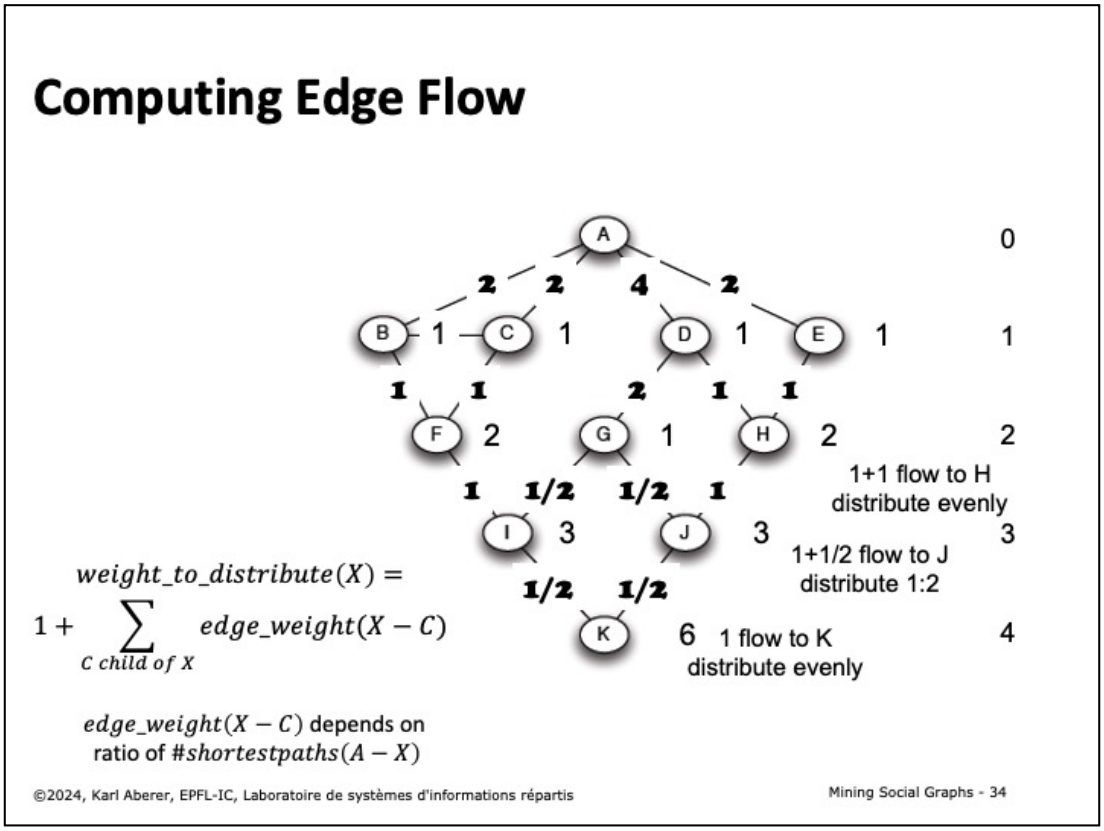
Computing Betweenness – Edge Flow

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 though an edge equals the betweenness value



In order to compute the betweenness values for edges, we have to aggregate the contributions of all paths between arbitrary pairs of nodes. To do this the following model is adopted. From each node a flow is emanating to each other node of the graph, which has a total volume of 1. This flow is evenly distributed among all shortest paths, that lead from the source node to the target node. Note that these different shortest paths may be overlapping, i.e., have common edges. Therefore edges that are part of different shortest paths will receive contributions from all the shortest paths passing through them. Finally, the flows will be aggregated for the flows emanating from all nodes, ot obtin the betweenness value of the edge.



Here we illustrate the second phase of the betweenness computation, computing the flow values related to one node, in the example node A. For each node one must consider the flow that is arriving at the node, plus the flows that are passing through the nodes to another target node. For computing the aggregate edge flows, the algorithm will start at the farthest node, in the example node K, which has no shortest paths emanating. Therefore, the only flow it receives is the one assigned to itself, i.e., a flow of 1. Since the node can be reached through different shortest paths, which are represented by the incoming edges from the nodes of the next higher level, this flow is distributed proportionally to the number of shortest paths leading to the parent nodes. In the case of node K, there exist 3 shortest paths leading to its two parent nodes. Therefore, the flows are evenly distributed. This results in flows of ½ for the incoming edges of node K.

For the nodes at the higher levels, the flows destined for the node are summed up with the flows destined for its descendant nodes that have been computed before. For example, node I has an outgoing flow of ½ plus a flow of 1 destined for itself. Therefore, it has to distribute a total flow of 3/2 over its incoming edges. This distribution must be done proportionally to the number of shortest paths arriving at its parent nodes. Since 2 shortest paths arrive at node F and 1 shortest path arrives at node G, the flow of 3/2 is split at a ratio of 2:1 among the two incoming edges of node I. The analogous reasoning is applied to the other nodes.

In general, each node distributes tribute a weight of $1 + \sum_{C\ child\ of\ X} edge\_weight(X - C)$ over the incoming edges, i.e., the flow of 1 arriving at the node plus the sum of the flows on the outgoing edges. The distribution is done according to the ratios of the shortest paths arriving at the parent nodes.

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 edge flows have been computed for all nodes, the resulting values are summed up. Since for each shortest path two flows have been generated, corresponding to the traversal of the path in the two directions, the final aggregate flow is divided by 2.

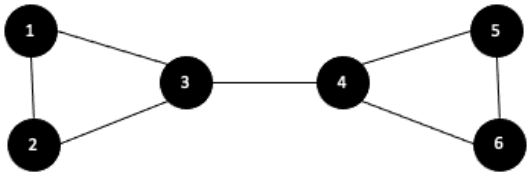
$\sigma_{xy}(v)$ of edge 3-4 is …

A. 16

B. 12

C. 9

D. 4



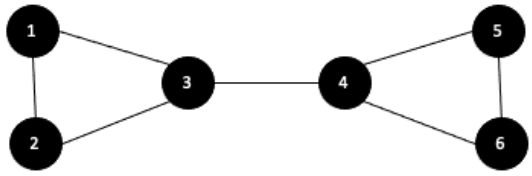
When computing path counts for node 1 with BFS, the count at 6 is …

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. Its major drawback is its scalability. The flow computation for one link has quadratic cost in the number of nodes, since it is 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.