TECHNISCHE UNIVERSITÄT DRESDEN

KNOWLEDGE GRAPHS

Lecture 13: Community detection

Markus Krötzsch

Knowledge-Based Systems

TU Dresden, 22nd Jan 2019

Review: Centrality

Centrality tries to identify the most important nodes in a network

- Many possible criteria (nodes from/to/through which something flows, emphasising longer or shorter distances travelled, possibly considering only certain paths, maybe using additional external criteria or weights)
- Putting importance on in-flow: in-degree centrality, eigenvector centrality, Katz centrality, PageRank

Simple recursive approximation for eigenvector-based measures:

- Start with any positive vector, and multiply powers of the matrix with it
- Pure form only meaningful for irreducible matrices (strongly connected graphs), but can be relaxed (Katz, PageRank)
- Typically converges quickly, and can be parallelised

Page vs. Katz

We can see strong parallels between the two algorithms:

- Both are all-paths criteria based on the number of paths leading to a node
- Both use a parameter to reduce the impact of long paths

Page vs. Katz

We can see strong parallels between the two algorithms:

- Both are all-paths criteria based on the number of paths leading to a node
- Both use a parameter to reduce the impact of long paths

The main difference is that Katz propagates the full (discounted) weight to all successors of a node, whereas PageRank (like eigenvector centrality) splits the weight among all successors:

- Katz centrality: "A node is important if it is highly linked or if it is linked from other important nodes."
- PageRank: "A node is important if it is highly linked or if it is linked from other important nodes that do not link to many other pages."

But maybe the most important difference is that one of the two has been famed as the secret of Google's success . . .

Shortest paths

Closeness

Previous centrality measures emphasised numbers of (highly valued) paths, and considered all available paths (including cycles).

In some applications, it is more meaningful to consider shortest paths.

Closeness

Previous centrality measures emphasised numbers of (highly valued) paths, and considered all available paths (including cycles).

In some applications, it is more meaningful to consider shortest paths.

In a (strongly) connected (di)graph with N nodes, we can calculate the average distance of a node i to all other nodes (excluding itself) as

$$\frac{\sum_{j\neq i}\mathsf{dist}(i,j)}{N-1}$$

where $\mathsf{dist}(i,j)$ is the length of the shortest path from i to j.

Closeness

Previous centrality measures emphasised numbers of (highly valued) paths, and considered all available paths (including cycles).

In some applications, it is more meaningful to consider shortest paths.

In a (strongly) connected (di)graph with N nodes, we can calculate the average distance of a node i to all other nodes (excluding itself) as

$$\frac{\sum_{j \neq i} \mathrm{dist}(i, j)}{N - 1}$$

where $\mathrm{dist}(i, j)$ is the length of the shortest path from i to j.

The corresponding centrality uses the inverse to assign higher values to nodes closer to the rest:

Definition 13.1: The closeness centrality of a node i in a strongly connected graph with N nodes is defined as $\frac{N-1}{\sum_{j \neq i} \mathrm{dist}(i,j)}$.

Closeness centrality

Limitations:

- Many networks have very small diameters, i.e., every node is rather close to every other node. This means that centrality scores are also close
- In particular, the most central node may not have a significant margin to other high-ranked nodes (which is different, e.g., in typical eigenvector results)
- Not defined for graphs that are not strongly connected (“infinite” distance to just one node would lead to centrality 0)

Closeness centrality

Limitations:

- Many networks have very small diameters, i.e., every node is rather close to every other node. This means that centrality scores are also close
- In particular, the most central node may not have a significant margin to other high-ranked nodes (which is different, e.g., in typical eigenvector results)
- Not defined for graphs that are not strongly connected ("infinite" distance to just one node would lead to centrality 0)

Possible solutions:

- Use closeness centrality on graphs where distance is more meaningful (e.g., closeness in a road network might indicate good locations for distribution centres)
- Restrict to largest strongly connected component (if very large and dominant) or assign a maximal N-step distance (rather than ∞) to unconnected nodes

Harmonic centrality

Another approach to solve the problem with disconnected graphs is to modify computation:

Definition 13.2: The harmonic centrality of a node i in a strongly connected graph with N nodes is defined as $\sum_{j\neq i} \frac{1}{(N-1)\mathsf{dist}(i,j)}$.

Note: Nodes without a connecting path can be considered to have infinite distance. This only affects one term of the sum, not the whole result.

Betweenness

Idea: Measure how relevant a vertex is for mediating flows along short paths

Example 13.3: In a communication network (e.g., the Internet), some nodes (routers) can spy on more traffic than others, since they are on the best connection between more users.

We will therefore define centrality as follows:

"A node is important if it is on the shortest path between many pairs of other nodes."

Betweenness centrality

Definition 13.4: In a graph with N nodes, the betweenness centrality of a node k is given by the sum

$$\frac{1}{(N-1)(N-2)} \sum_{i \neq j, i \neq k, j \neq k} \frac{\mathsf{sp}_{ij}(k)}{\mathsf{sp}_{ij}}$$

where sp_{ij} is the number of shortest paths from i to j, and $\mathsf{sp}_{ij}(k)$ is the number of such paths through k.

Notes:

- There might be several, partly overlapping shortest paths between i and j.
- $(N-1)(N-2)$ is the number of pairs of distinct nodes i and j that are both unequal to k; this is a normalisation factor.
- High betweenness scores are achieved by the central nodes of star-shaped graphs; low scores occur, e.g., for dead ends (which are on no shortest path at all)

Betweenness: uses and limits

Betweenness is robustly applicable, but different from most other metrics, since nodes with high betweenness might be:

- connected to only few other nodes,
- far from more densely connected areas, and
- not directly or indirectly related to nodes of similarly high importance

$\leadsto$ application scenarios for betweenness are frequent, but completely different from scenarios where other metrics might be used.

Betweenness: uses and limits

Betweenness is robustly applicable, but different from most other metrics, since nodes with high betweenness might be:

- connected to only few other nodes,
- far from more densely connected areas, and
- not directly or indirectly related to nodes of similarly high importance

$\leadsto$ application scenarios for betweenness are frequent, but completely different from scenarios where other metrics might be used.

Implementation: this measure is calculated in the direct (laborious) way, iterating over all pairs of nodes and all shortest paths to compute the sum. This is usually cubic time, but can go down to quadratic for sparse graphs.

Overview of centrality criteria

- Degree centrality: “A node is important if it is highly linked.”
- Eigenvalue centrality: “A node is important if it is linked from other important nodes.”
- Katz centrality: “A node is important if it is highly linked or if it is linked from other important nodes.”
- PageRank: “A node is important if it is highly linked or if it is linked from other important nodes that do not link many other pages.”
- Closeness centrality: “A node is important if it is close to many other nodes.”
- Betweenness centrality: “A node is important if it is on the shortest path between many pairs of other nodes.”

Finding communities

Communities in networks

Intuition: Many graphs are not uniform but can be viewed as consisting of several (possibly overlapping) communities

- Nodes of the same community should be more likely to be connected than nodes in different communities.
- If communities are disjoint, then the graph consists of relatively dense graphs with only relatively few connections in between

Example 13.5: The term comes from social network analysis. Indeed, people that share goals or interests often form communities.

Communities in networks

Intuition: Many graphs are not uniform but can be viewed as consisting of several (possibly overlapping) communities

- Nodes of the same community should be more likely to be connected than nodes in different communities.
- If communities are disjoint, then the graph consists of relatively dense graphs with only relatively few connections in between

Example 13.5: The term comes from social network analysis. Indeed, people that share goals or interests often form communities.

Motivation:

- Community structures can reveal insights into how networks emerged
- Characteristics of one community can be very different from the characteristics of another, or of the overall graph

$\leadsto$ knowing some community structure improves our understanding and makes many algorithms more effective

The method of Girvan and Newman

Idea:

- High betweenness is typical for nodes that are "between" communities
- Betweenness can also be computed for edges

$\leadsto$ Decompose networks into communities by removing edges of high betweenness

The method of Girvan and Newman

Idea:

- High betweenness is typical for nodes that are “between” communities
- Betweenness can also be computed for edges

$\rightsquigarrow$ Decompose networks into communities by removing edges of high betweenness

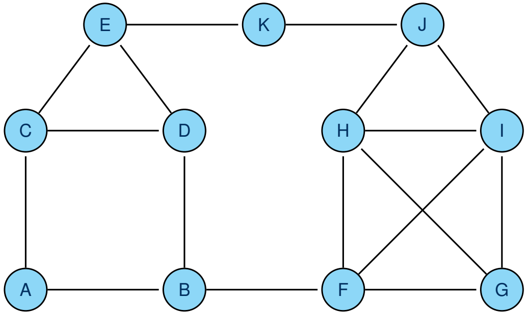
Girvan-Newman Algorithm:

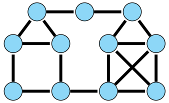
```
Repeat
    Compute betweenness for all edges
    Remove edge with largest betweenness value
    Detect connected components of remaining graph (“communities”)
Until “enough communities have been found”
```

The GN algorithm actually produces a hierarchical clustering of a graph:
it decomposes the graph into smaller and smaller clusters, down to single nodes.

GN algorithm: Example

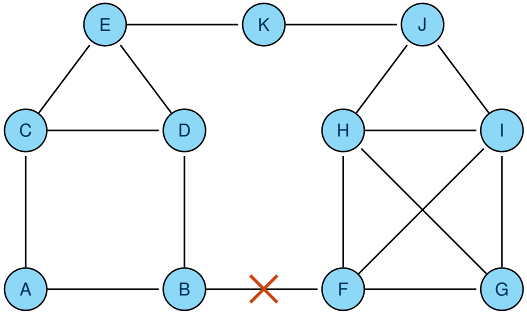
We consider an undirected network:

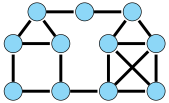




GN algorithm: Example

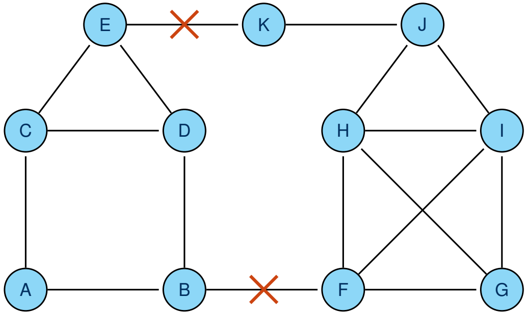
We consider an undirected network:

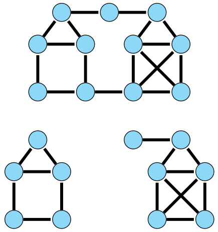




GN algorithm: Example

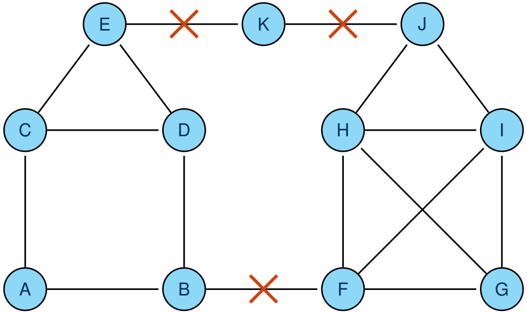
We consider an undirected network:

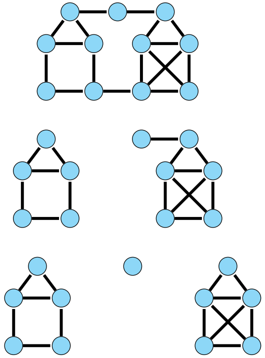




GN algorithm: Example

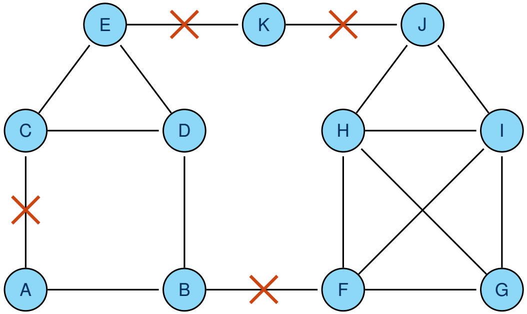
We consider an undirected network:

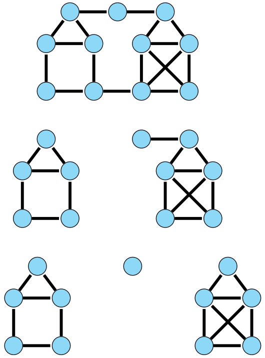




GN algorithm: Example

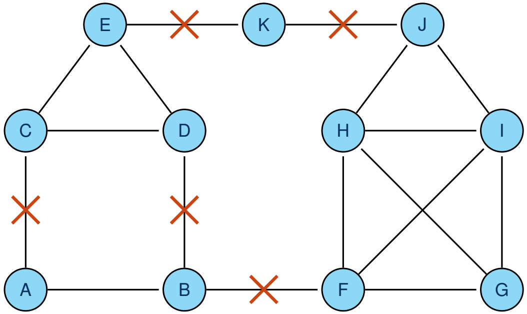
We consider an undirected network:

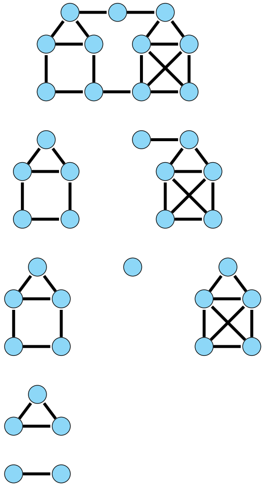




GN algorithm: Example

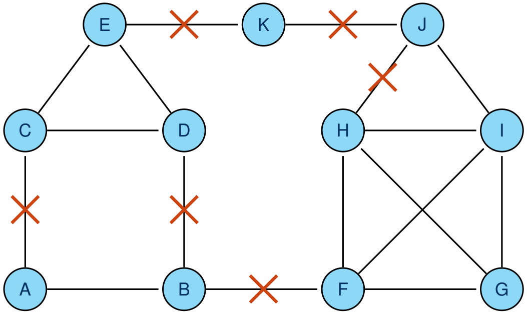
We consider an undirected network:

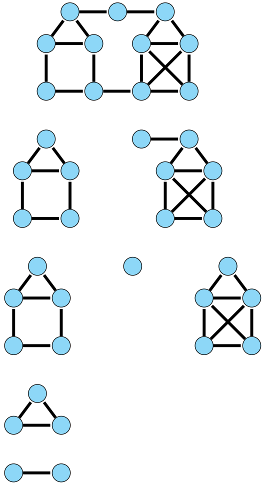




GN algorithm: Example

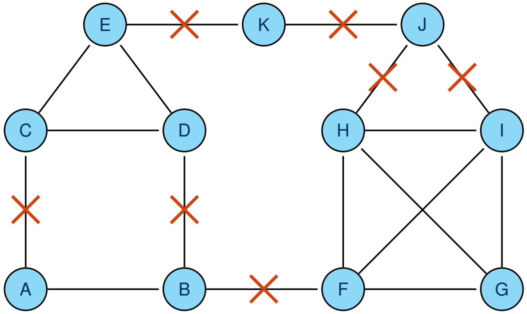
We consider an undirected network:

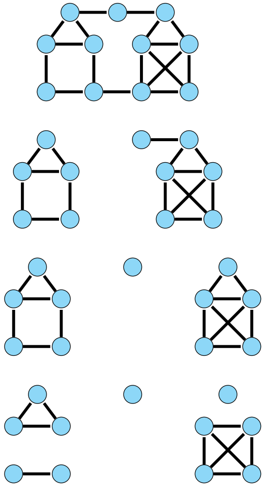




GN algorithm: Example

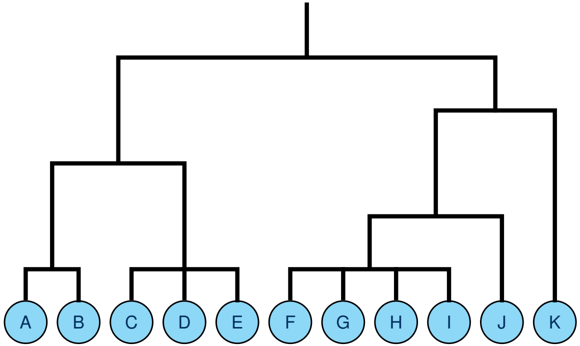
We consider an undirected network:

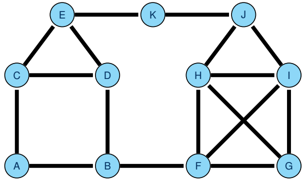




Hierarchical clustering

Like any hierarchical clustering, the output of betweenness-based clustering can be visualised in a dendrogram that shows how connected components (communities) change when removing edges:





The possible clusters can be read off this tree structure (at the desired level). The edge-removal process can be continued until each community has only one member.

Approximating betweenness

The computation of betweenness for all edges is time-consuming:

- For all possible source vertices s
- and for all possible target vertices t:
- find all shortest paths from s to t
- and add their relative contributions to the edges.

$\leadsto$ requires n^2 shortest-path computations – way too inefficient

Approximating betweenness

The computation of betweenness for all edges is time-consuming:

- For all possible source vertices s
- and for all possible target vertices t:
- find all shortest paths from s to t
- and add their relative contributions to the edges.

$\leadsto$ requires n^2 shortest-path computations – way too inefficient

Better algorithms exist:

- For all possible source vertices s
- find all shortest paths from s to any other t using breadth-first search
- and add their relative contributions to the edges.

$\leadsto$ still too slow on large graphs

Approximating betweenness

The computation of betweenness for all edges is time-consuming:

- For all possible source vertices s
- and for all possible target vertices t:
- find all shortest paths from s to t
- and add their relative contributions to the edges.

$\leadsto$ requires n^2 shortest-path computations – way too inefficient

Better algorithms exist:

- For all possible source vertices s
- find all shortest paths from s to any other t using breadth-first search
- and add their relative contributions to the edges.

$\leadsto$ still too slow on large graphs

Approximate solutions may help:

- Select a subset of vertices by random sampling, and
- perform the breadth-first search only from those sources

When to stop?

The GN algorithm does not define at which level the clustering is most meaningful.

When to stop?

The GN algorithm does not define at which level the clustering is most meaningful.

Possible solutions:

- Define a target number of communities and stop as soon as there are at least that many disconnected components.
  $\leadsto$ requires prior knowledge on network structure
- Define a maximum size for communities and stop as soon as all communities are below this size.
  $\leadsto$ requires a very specific network structure and prior knowledge to make sense
- Define a measurement for community quality and stop as soon as the produced communities are good enough.
  $\leadsto$ requires a meaningful quality criterion

Modularity

Goal: Given a hypothetical community partitioning, estimate its quality.[1]

Idea: Communities should be more densely connected internally than random graphs

Modularity: “# edges in communities – expected # if edges were random”

[1] We will restrict to undirected graphs here.

Modularity

Goal: Given a hypothetical community partitioning, estimate its quality.[1]

Idea: Communities should be more densely connected internally than random graphs

Modularity: "# edges in communities – expected # if edges were random"

Main design decision: What model to use to estimate "random" edge distribution?

[1] We will restrict to undirected graphs here.

Modularity

Goal: Given a hypothetical community partitioning, estimate its quality.[1]

Idea: Communities should be more densely connected internally than random graphs

Modularity: "# edges in communities – expected # if edges were random"

Main design decision: What model to use to estimate "random" edge distribution?

- Easiest approach: consider a random graph with same number n of vertices and m of edges as the original graph – expect $m\frac{\ell^2}{n^2}$ edges in a community of ℓ nodes

[1] We will restrict to undirected graphs here.

Modularity

Goal: Given a hypothetical community partitioning, estimate its quality.[1]

Idea: Communities should be more densely connected internally than random graphs

Modularity: "# edges in communities – expected # if edges were random"

Main design decision: What model to use to estimate "random" edge distribution?

- Easiest approach: consider a random graph with same number n of vertices and m of edges as the original graph – expect $m\frac{\ell^2}{n^2}$ edges in a community of ℓ nodes
- Problem: random graphs are very homogeneous, with any node having the same expected number of neighbours, while many real-world networks have very uneven degree distributions

$\leadsto$ use a random model that preserves node degrees

[1] We will restrict to undirected graphs here.

Degree-preserving random graphs

Intuitive approach: start with given network, cut each edge into two stubs, and glue the stubs back together randomly (possibly even in self-loops)
$\leadsto$ degree of each node remains the same, but edges connect random nodes

Degree-preserving random graphs

Intuitive approach: start with given network, cut each edge into two stubs, and glue the stubs back together randomly (possibly even in self-loops)
$\rightsquigarrow$ degree of each node remains the same, but edges connect random nodes

Some calculations: for a graph with m edges

- Probability of two specific stubs being glued together: $\frac{2m}{(2m)^2} = \frac{1}{2m}$
- Possible ways of connecting two nodes i and j: degree$(i) \cdot$ degree(j)
  $\rightsquigarrow$ expected edges between two nodes i and j: $\frac{\text{degree}(i)\,\text{degree}(j)}{2m}$
- Actual connections between two nodes i and j: Conn$(i, j) \in \{0, 1\}$

Degree-preserving random graphs

Intuitive approach: start with given network, cut each edge into two stubs, and glue the stubs back together randomly (possibly even in self-loops)
$\rightsquigarrow$ degree of each node remains the same, but edges connect random nodes

Some calculations: for a graph with m edges

- Probability of two specific stubs being glued together: $\frac{2m}{(2m)^2} = \frac{1}{2m}$
- Possible ways of connecting two nodes i and j: degree$(i) \cdot$ degree(j)
  $\rightsquigarrow$ expected edges between two nodes i and j: $\frac{\text{degree}(i)\,\text{degree}(j)}{2m}$
- Actual connections between two nodes i and j: Conn$(i,j) \in \{0, 1\}$

The modularity Q is defined as follows:

$$Q = \frac{1}{4m} \sum_{\text{Community } C} \sum_{i,j \in C} \left(\text{Conn}(i,j) - \frac{\text{degree}(i)\,\text{degree}(j)}{2m} \right)$$

Note: The factor $\frac{1}{4m}$ ensures that $-1 \leq Q \leq 1$.

Using modularity in the GN algorithm

$$Q = \frac{1}{4m} \sum_{\text{Community } C} \sum_{i,j \in C} \left(\text{Conn}(i,j) - \frac{\text{degree}(i)\,\text{degree}(j)}{2m} \right)$$

Values $Q > 0$ indicate above random edge density inside of communities, with higher values being better

Approach:

- Compute a complete clustering using the GN algorithm
- Select the partitioning with the highest Q value

Modularity-based community detection

The GN algorithm is not the only algorithm that can make use of modularity:

- One can also define the search for a modularity-maximising partitioning as an optimisation problem
- This intractable problem can be heuristically solved in many ways (see, e.g., the “Louvain method”)

Modularity-based community detection

The GN algorithm is not the only algorithm that can make use of modularity:

- One can also define the search for a modularity-maximising partitioning as an optimisation problem
- This intractable problem can be heuristically solved in many ways (see, e.g., the “Louvain method”)

However, such approaches have some inherent limitations:

- Typical networks grow much faster than the maximal degree of their nodes
- At some size, the term $\frac{\text{degree}(i)\,\text{degree}(j)}{2m}$ approaches 0

$\leadsto$ any edge between two groups of nodes is considered a sign for community structure

Modularity-based approaches therefore cannot detect small communities in large networks (merging them into any community they are connected with will always lead to a better values).

Moreover, fast approximate approaches can be far off the actual optimum.

Further approaches to community detection

Further approaches to community detection (1)

Minimum Cuts: cut graph into disconnected components by removing edges

- Prefer cuts that (1) cut only few edges and (2) lead to components of similar size (the approach of assigning a to-be-minimised value to a cut may vary)
- Several efficient algorithms are available
- Good for partitioning network into “local” neighbourhoods by cutting communication bottlenecks (e.g., data placement in distributed computing), less suitable for community detection

Graph partitioning is mostly used in slightly different application areas.

Further approaches to community detection (2)

Bi-Cliques ("Trawling"): discover communities starting from large complete bi-partite subgraphs

- On general graphs, the approach works as follows:
  (1) Split graph into two equal-sized vertex sets
  (2) Look for sub-sets where all vertices on the left are connected to all vertices on the right (mathematical background: in sufficiently dense communities, large bi-cliques must exist)
  (3) Grow communities around the vertices by adding highly linked vertices
- Efficient computation uses techniques from frequent itemset mining to search such "bi-cliques"
- Good for finding also small communities; applicable also to k-partite graphs

Very common approach in social network analysis; can be applied to rather large networks.

Further approaches to community detection (3)

Statistical inference: try to find a model that explains the observed data

- Consider some parametrised graph generator (“the model”), and try to find the parameter values that are most likely to lead to the given graph
- Generally hard to solve and depending on model; many algorithms exists, e.g., search methods for continuous parameter fitting (e.g., gradient descent)
- Capable of finding overlapping communities; might lead to even deeper insights into the nature of the graph than mere community detection

More complex but potentially more insightful and of greater predictive value

Summary

Closeness and betweenness centrality evaluate the shortest paths for rather different results

The Girvan and Newman algorithm exploits (edge) betweenness for clustering networks into communities

Modularity is an imperfect measure for the quality of a hypothesised community structure

What's next?

- More network analysis
- Summary
- Examinations