Universiteit Utrecht

[Faculty of Science
Information and Computing Sciences]

Community Detection

Network Science 2019/2020
Johan M. M. van Rooij
Erik Jan van Leeuwen

The speaker: Johan van Rooij

- Assistant Professor (Tuesday only)
  - Software Project (lectures)
  - Algorithms and Networks
  - Network science (practical part)
  - Master Thesis supervision
- Senior Data Scientist in industry (rest of the week)

VIQTOR DAVIS
DATA CRAFTSMANSHIP

NEDERLANDSE DATA SCIENCE PRIJZEN

VALID
STAY AHEAD

CQM
Consultants in Quantitative Methods

Universiteit Utrecht

[Faculty of Science
Information and Computing Sciences]

Three hours of lectures on a row

- Community Detection
  - Introduction to basic concepts
  - Gives you some idea of what topic to choose, if you want to do a practical programming project.
- Experimental term paper
  - What we expect of you
  - Brief overview of the different topics to choose.
  - Some guidelines on how to do algorithmics experiments properly: hypotheses, data gathering, testing.
- Literature term paper (short lecture)
  - Brief overview of the different topics to choose.

Community Detection - Network Science

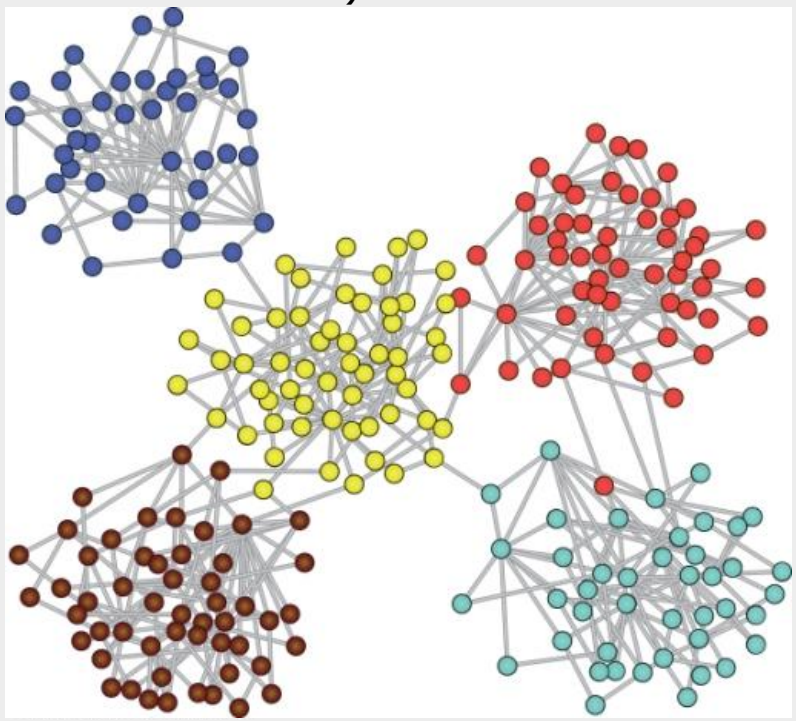
COMMUNITY DETECTION

We study networks

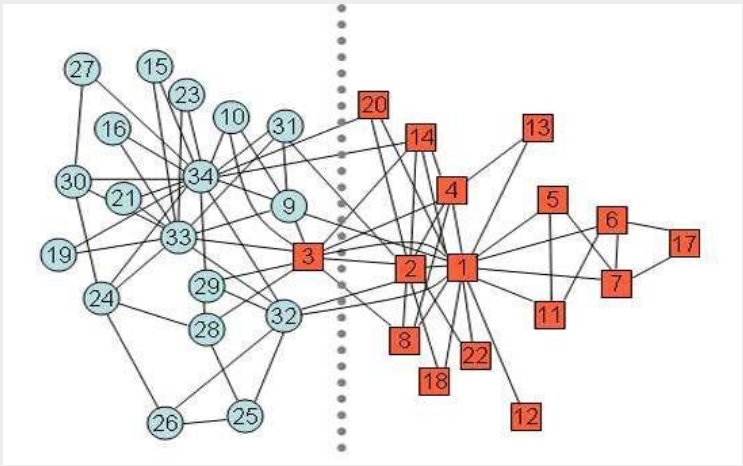
- We study networks:
  - Because we want to understand the things they represent.
  - How does it function? What can we learn from it?
- Social networks
  - Marketing
  - Disease control
  - Structure of a society
  - …
- Internet
  - Flow of data traffic
  - How to increase performance
- …

Community structure

- Community structure does not only apply to social networks.
  - Biology: identify groups of molecules in functional modules.
  - Road networks: preprocessing for fast routing (see Algorithms & Networks)



From: GitHub, Benedek Rozemberczki. Awesome Community Detection Research Papers



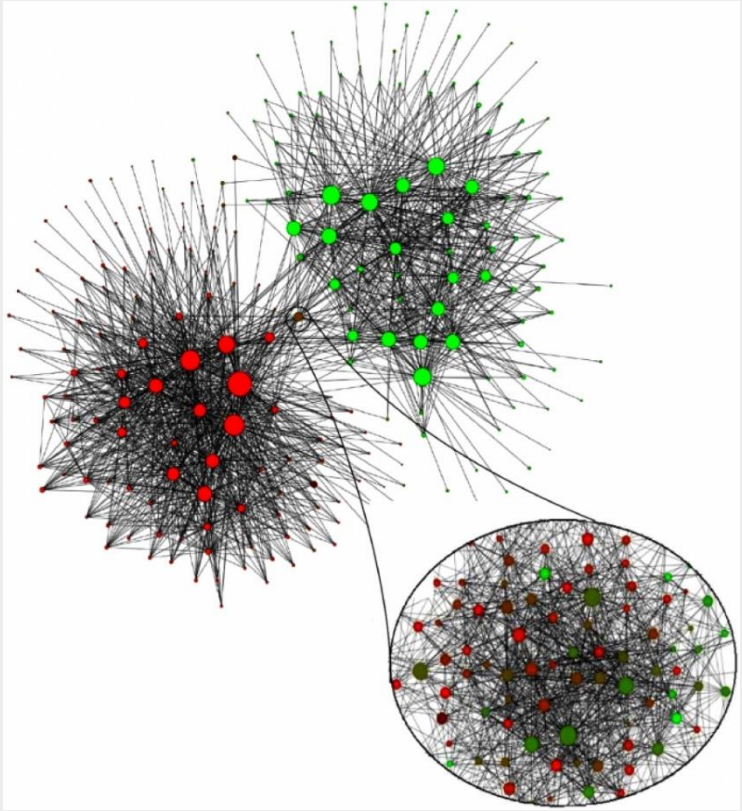
The Karate Club Network of Zachary with one misclassified node

Example: Belgian phone network (2007)

- **Vertices**: phones
- **Edges**: regular calls node between both phones

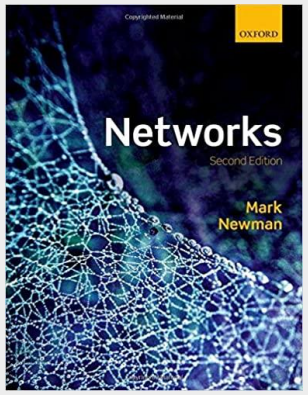
- We see two clusters
  - (And a small group of phones highly connected to both clusters).

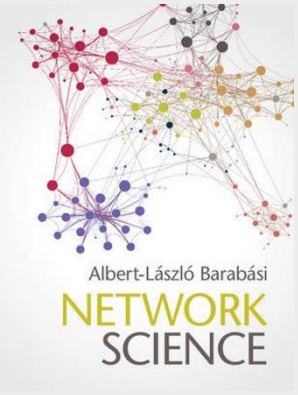
- What does this say about Belgium?



Literature and software/code

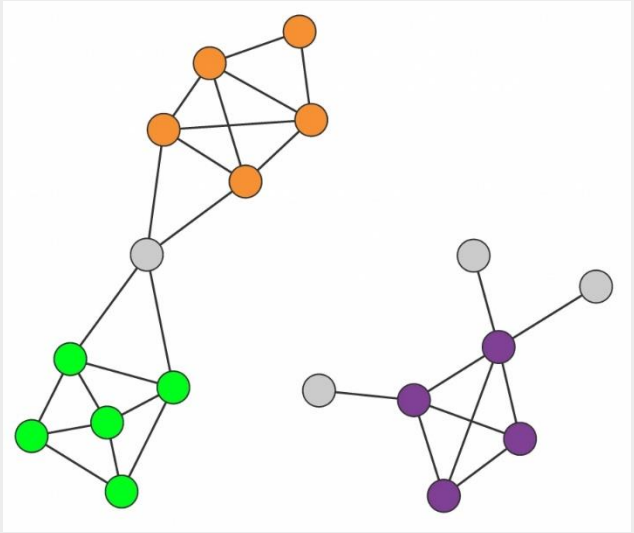
- Most of this lecture is based on the books for this course.





- You can also find great stuff here:
  - "Community detection in networks: a user guide", Santo Fortunato and Darko Hric. https://arxiv.org/abs/1608.00163
  - "Awesome Community Detection Research Papers", Benedek Rozemberczki, GitHub. Overview of papers (with code). https://github.com/benedekrozemberczki/awesome-community-detection
  - The NetworkX python package (https://networkx.github.io/) and extensions such as https://python-louvain.readthedocs.io/en/latest/

What is a community?

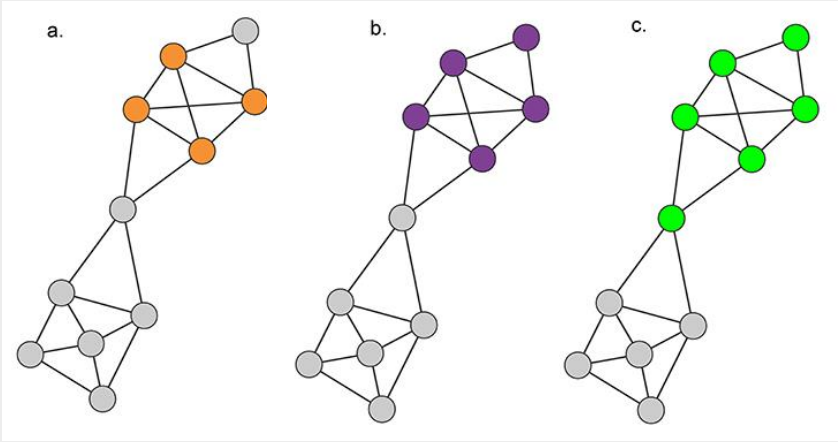


- This is a non-trivial question!
  - Visually you have some idea…
  - However, defining this in terms of vertices and edges is hard.
- Classically: a community is a **maximum clique**.
  - Clique: set of vertices with an edge between each pair.
  - Maximum clique: clique that is not a subclique of another clique.
- This notion, however, is often not very useful.
  - Graphs on this slide: only the purple vertices form a clique.
  - Belgian phone networks: communities are not cliques.

What a community should be?

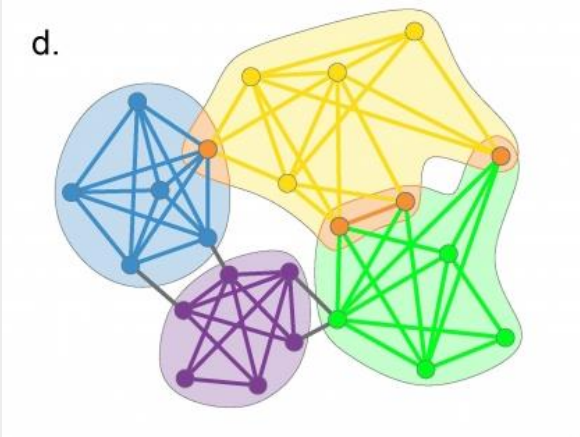
- A community C in G=(V,E) should at least be:
  - **Connected**.
  - **Dense**: more edges between vertices in C that between vertices in C and V\C.

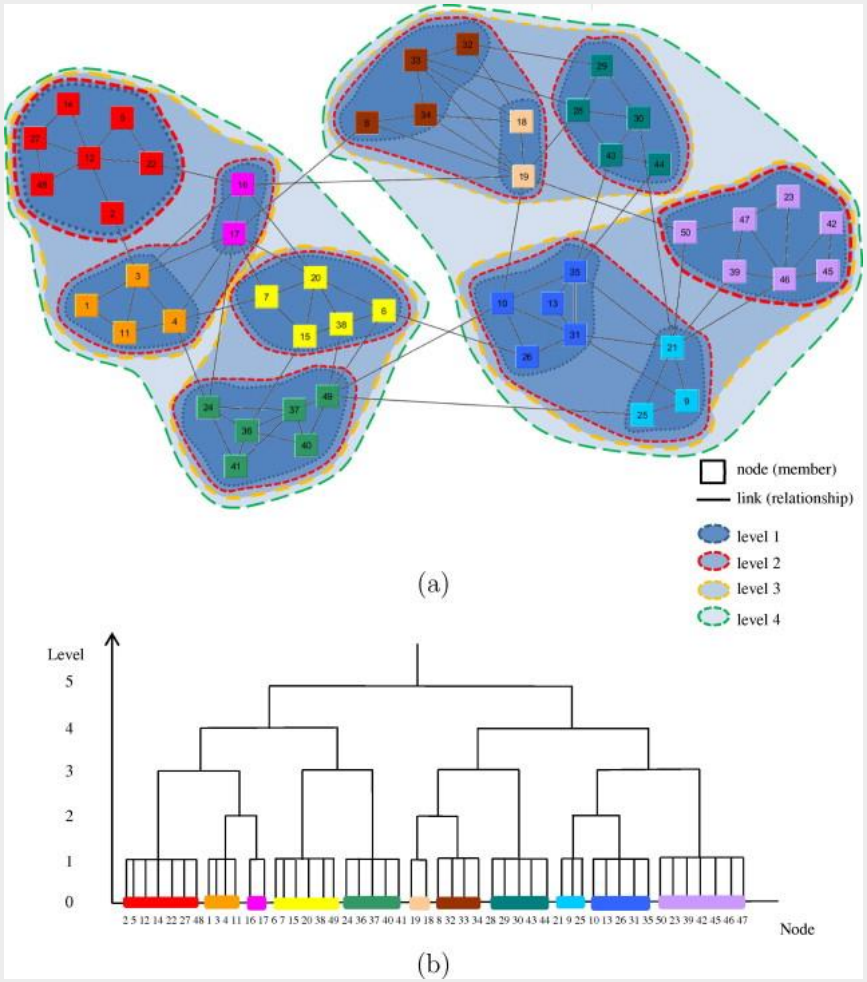
- **Clique** (a).
- **Strong community** (b):
  - Each vertex in C has more neighbours in C than in V\C.
- **Weak community** (c):
  - $\sum_{v \in C} |N(v) \cap C| > \sum_{v \in C} |N(v) \backslash C|$



Much freedom and variants

- Hierarchical communities?
  - Hierarchical clustering
- Overlapping communities…
  - E.g. communities defined by edges.
- Should all vertices be in a community?





Community Detection - Network Science

MODULARITY

Modularity
an often used measure of community structure

- Given a graph, if we:
  - Fix the degrees of the vertices of a graph.
  - Connect the endpoints by m edges by repeatedly choosing two endpoints uniformly at random.
- Then, we would expect that:
  - There is no community structure in G.
- If we have a partition of the vertices into communities:
  - We expect the number of edges between vertices in the same community to be (much) higher than in the above random graph.
  - This can be made into a measure of how good the community structure is.
- Let us make this more precise…

The expected number of edges between two vertices

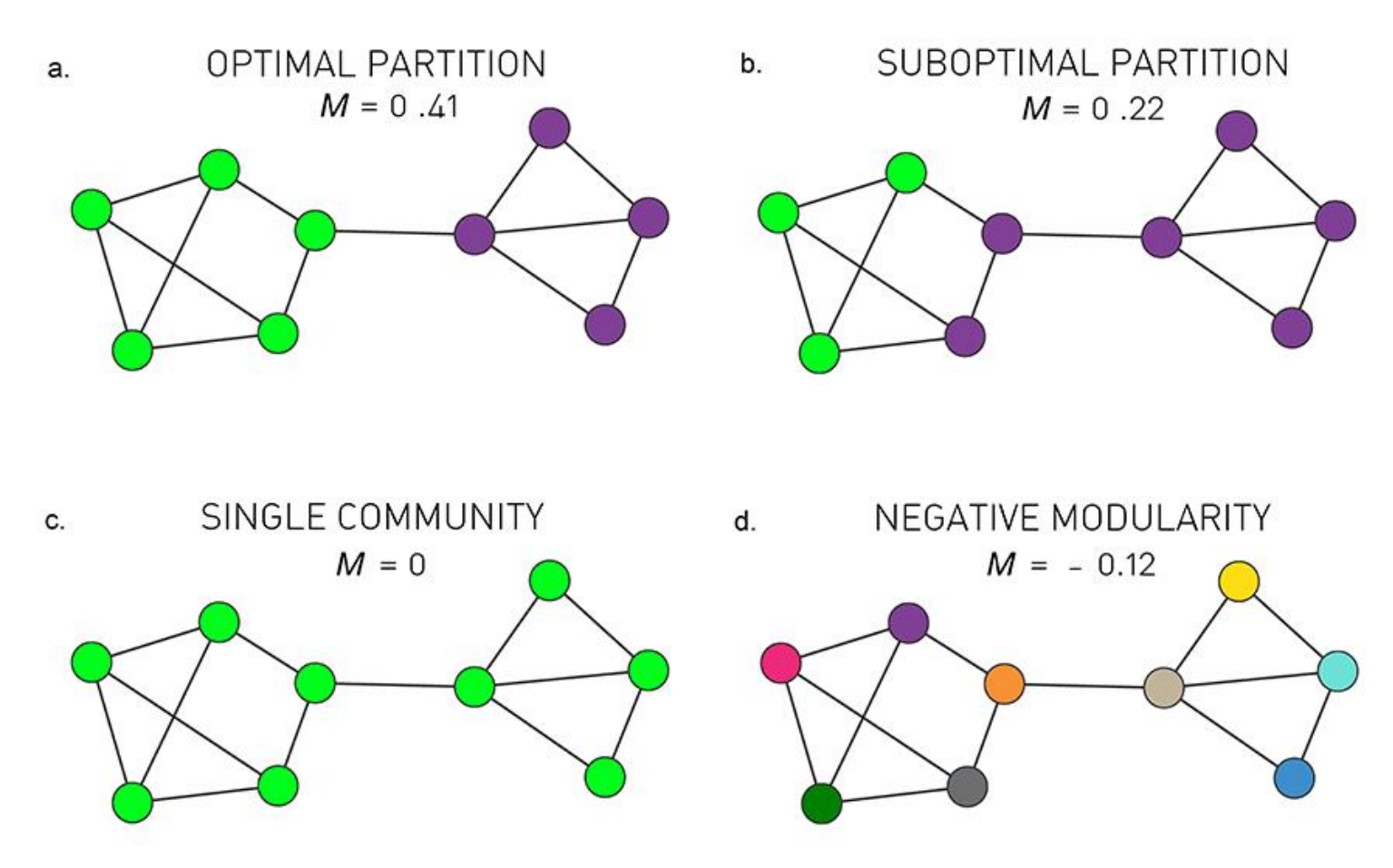
- Given a graph, if we:
  - Fix the degrees of the vertices of a graph.
  - Connect the endpoints by m edges by repeatedly choosing two endpoints uniformly at random.

- Expected number of edges between two vertices i and j:
  - For one fixed endpoint of vertex i, any other of the $2m-1$ endpoints are equally likely to connect to (allowing self-loops).
  - Probability that it connects to j: $\frac{k_j}{2m-1}$, where k_j the degree of j.
  - Expected number of edges between i and j: $\frac{k_i k_j}{2m-1}$
  - Often the -1 is omitted because for large m the difference is small.

Modularity:
difference to random connections

- Modularity: $$M = \frac{1}{2m} \sum_{c \in C} \sum_{i,j \in c} \left(A_{ij} - \frac{k_i k_j}{2m} \right)$$
- Here, we
  - Sum over all communities.
  - A_{ij} is the incidence matrix: 1 if an edge 0 otherwise.
  - So we sum over difference between nr of edges and expected nr of edges in the random model.
  - $\frac{1}{2m}$ is to normalise the result (maximum of modularity is 1).
- Maximum modularity M for a graph G:
  - M>0, there are potential communities;
  - M=0, equal to random model;
  - M<0, no community structure.

Modularity:
some examples



Find the community structure with maximum modularity

- Finding the community structure with maximum modularity is NP-hard.
  - With c communities, trying all c^n options is too slow for even moderate size graphs.
  - Methods from optimisation literature (simulated annealing, tabu search, genetic algorithms, etc.) often too slow on large graphs.
- Many practical algorithms known:
  - Greedy algorithm (e.g. the one by Newman, see Barabási book)
  - *Simple node moving (see Newman book)*
  - Spectral decomposition (see Newman book)
  - *Louvain algorithm (see Newman book) – very popular!*
- All these algorithms are heuristics.
  - No guarantees on optimal solutions (often not on large graphs).
  - Fast run times, but significant differences on large graphs.

Simple node moving
algorithm for partition the graph in two communities

Simple node moving algorithm:

1. Randomly partition the graph into two equal size sets.
2. Initialise a set X with all vertices.
    1. For each vertex in X, compute the effect on the modularity by moving it to the other partition.
    2. Choose the vertex from X with the highest increase/lowest decrease of modularity: remove it from X, change its partition.
    3. Repeat the above until X is empty, and keep track of the best solution found.
3. Repeat step 2 starting with the best solution found in last round until no improvement.
4. Return best solution found in the last round.

On simple node moving…

- Some smart evaluation tricks (see Newman book), allow:
  - Computing delta to modularity for all vertices in O(m+n) time.
  - Hence a round costs O(n(m+n)) time.
- However, run time analysis is not the full story:
  - It is a heuristic! Not an exact optimisation algorithm.
  - Comparisons should include experimental results and run times.
- In all a reasonable algorithm.
  - Simple to implement and understand.
  - Relatively slow compared to Louvain.
  - Results have reasonable quality.

Louvain

algorithm for partition the graph in multiple communities

Louvain

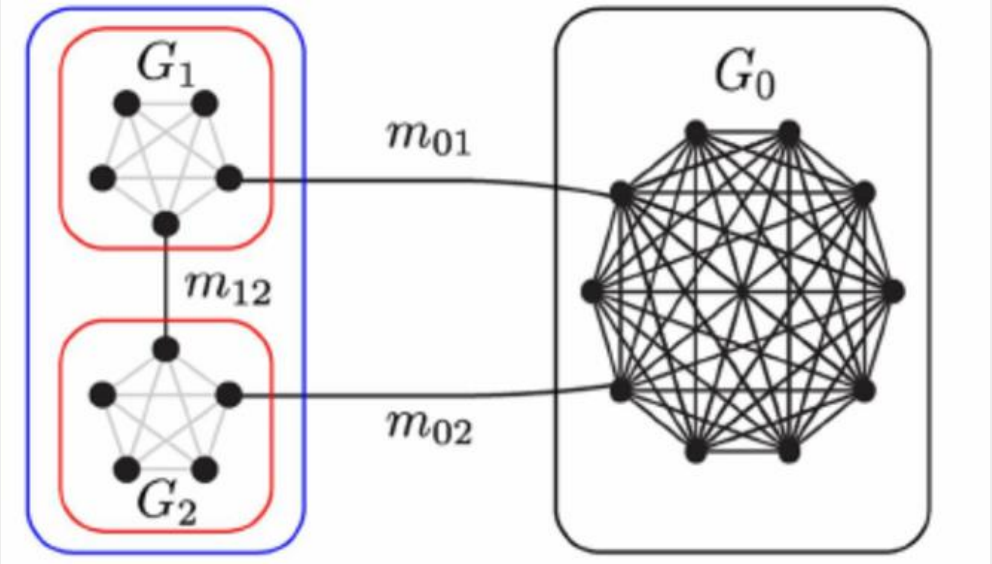
1. Initially let every vertex be it's own partition/community.
2. For each vertex in the graph:
    1. Compute the effect on the modularity by moving it to a neighbouring partition.
    2. Choose the partition with the highest increase of modularity (otherwise no move) and move it to the corresponding partition.
3. Repeat the above until no improvement.
4. Return to step 2, only now considering moving an entire group of vertices instead of separate vertices.
    - Groups of vertices: communities from the previous round.
5. Stop at step 4 when a round results in no changes.

On Louvain…

- Using the same tricks step 2 can be implemented in O(m+n) time.
- The question is how many repetitions occur, and how often we shrink the graph.
  - Without a more thorough theoretical analysis, experiments show that in practice it runs in roughly O(n log n) if m is O(n).
- Louvain directly optimises the multi-community problem.
- The simple node moving algorithm only works for the 2-community version.
  - Repeated subdivision of communities can be used here.
  - However, this can never recover from one wrong subdivision.

Last remarks on modularity

- Modularity is just one measure of community structure.
  - Many others exist.
- A know downside of modularity is called the ‘resolution limit’.
  - Small communities compared to the size of the full graph cannot be found.
  - That is, the modularity is higher if small communities are merged together.

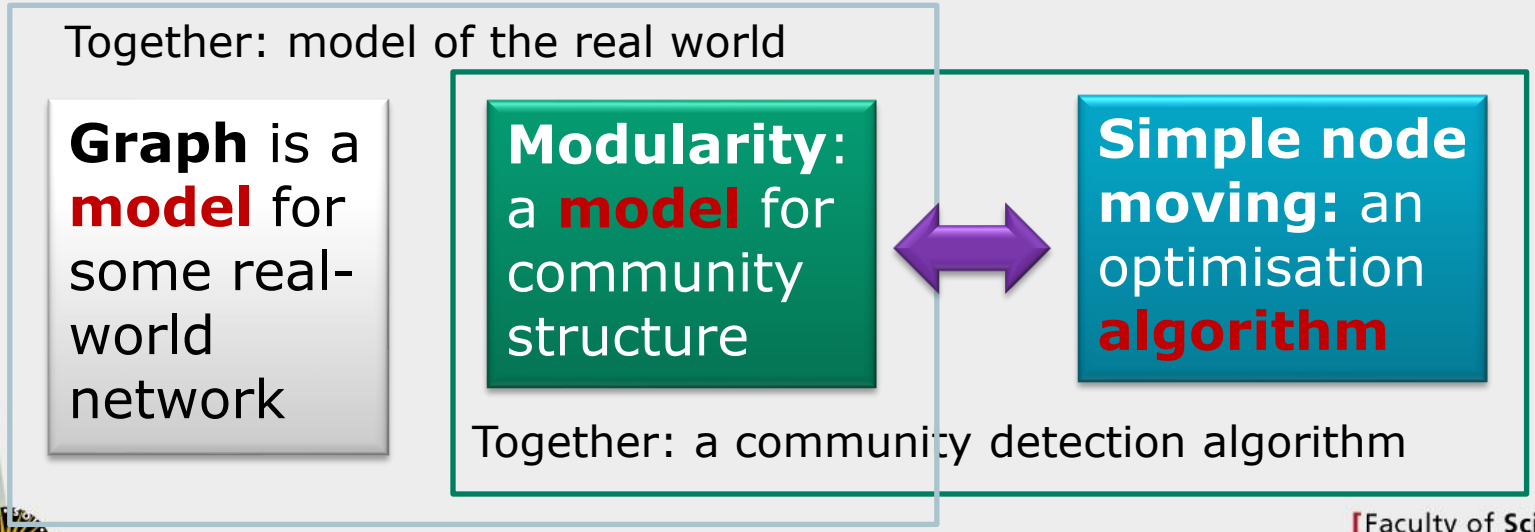


Community Detection - Network Science

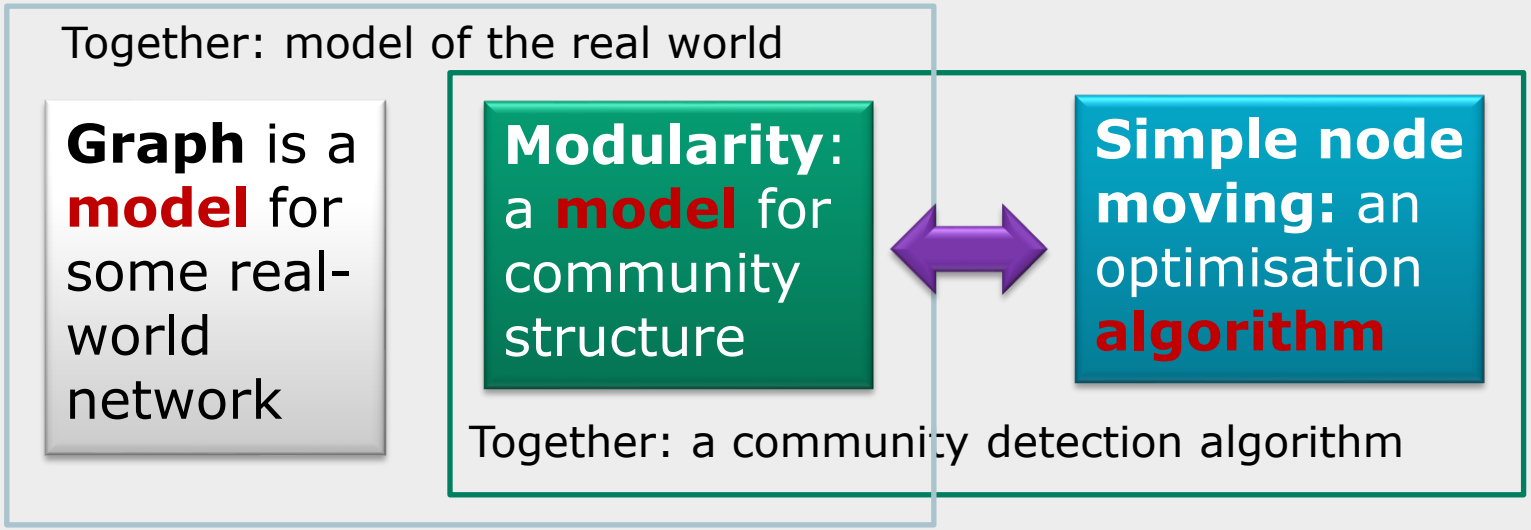
SHORT INTERMEZZO: MODELS AND ALGORITHMS

Model vs Algorithm

- Wikipedia:
  - A mathematical model is a description of a system using mathematical concepts and language.
  - An algorithm is a finite sequence of well-defined, computer-implementable instructions, typically to solve a class of problems or to perform a computation.



What does the result say?



- Any community in the real world:
  - How well does the graph contain information on it?
  - Is modularity the measure to use here?
  - How well does the heuristic algorithm perform?
  - Or is suboptimal optimisation a feature?
- Also, often you can replace the inner model or inner algorithm without changing the others.

Community Detection - Network Science

STATISTICAL-INFERENCE-BASED METHODS

Statistical-inference-based methods

- Use statistical inference to find community structure.

1. Specify a random graph model that includes a sense of communities.
   - Here, you can add knowledge about the domain the graph describes.
2. Find the values for the parameters of the model that fit best to the data (the graph)?
   - Fitting though statistical methods such as maximum likelihood estimation.

- We give an example in the next series of slides.
  - Warning: the presentation is based on the Newman book, however, it is also simplified to omit even more tedious formulas.

A random graph model
a slight modification to the degree-corrected stochastic block model

- Consider a random graph model with parameters:
  - Number of edges m.
  - Expected degrees for the vertices k_i.
  - Group structure G with group g_i for vertex i.
  - A $q \times q$ symmetric matrix P with probabilities p_{rs}.
- Graphs are generated by:
  1. Repeat 2-4 below m times:
  2. Pick a vertex i at random with probabilities $\frac{k_i}{2m}$.
  3. Pick a vertex j at random with probabilities $p_{g_i g_j} k_j$.
  4. Add edge (i, j).
- Note that this allows double edges and self loops!
  - Also note that line 3 puts constraints on the p_{rs}.

Probability on having an edge between two vertices

- For distinct vertices i and j, the probability of having k edges between them is Binomially distributed:

$$P(k\ edges\ i-j) = \binom{m}{k} p^k (1-p)^{m-k}$$

  - Where $p = 2\frac{p_{g_i g_j} k_i k_j}{2m} = \frac{p_{g_i g_j} k_i k_j}{m}$ if $i \neq j$, and $p = \frac{p_{g_i g_j} k_i k_j}{2m}$ if $i = j$.
  - Note that double edges are rare in this model, especially for large m (the probabilities are almost zero for $k > 1$).

- Computing with binomials can be cumbersome, so we use that for large m this converges to a Poisson distribution.

$$P(k\ edges\ i-j) = \frac{\lambda^k}{k!} e^{-\lambda}$$

  - Where $\lambda = mp$, i.e., $\lambda = p_{g_i g_j} k_i k_j$ if $i \neq j$, and $\lambda = \frac{p_{g_i g_j} k_i k_j}{2}$ if $i = j$.

Maximum likelihood estimation

- We now have:

$$P(A|P,G) = \prod_{i<j} P(X_{p_{g_i g_j} k_i k_j} = A_{ij}) \times \prod_i P(X_{\frac{p_{g_i g_i} k_i k_i}{2}} = \frac{A_{ii}}{2})$$

  - Where X_λ is Poisson distributed with parameter λ.
  - A is the incidence matrix of the graph.
- Maximum likelihood estimation:
  - Take $L(P,G|A) = P(A|P,G)$, the likelihood of parameters P, G given the data (the graph) A.
  - The parameters p_{rs}, k_i and g_i can be chosen such that $L(P,G|A)$ is maximum.
    - To simplify, we replace the expected degrees by the actual degrees of the graph fixing the k_i (these have maximum likelihood).
  - One usually maximises the log of the likelihood, which has the same maximum as it is a monotone function.

Working out the math

Note that the A_{ij} and k_i are constaints

$$L(P,G|A) = \prod_{i<j} P(X_{p_{g_i g_j} k_i k_j} = A_{ij}) \times \prod_i P\left(X_{\frac{p_{g_i g_i} k_i k_i}{2}} = \frac{A_{ii}}{2}\right)$$

$$= \prod_{i<j} \frac{\left(p_{g_i g_j} k_i k_j\right)^{A_{ij}}}{A_{ij}!} e^{-p_{g_i g_j} k_i k_j} \times \prod_i \frac{\left(p_{g_i g_i} k_i k_i\right)^{A_{ii}/2}}{(A_{ii}/2)!} e^{-p_{g_i g_i} k_i k_i}$$

$$\log\left(L(P,G|A)\right) = \sum_{i<j} \left(A_{ij} \log\left[p_{g_i g_j}\right] - p_{g_i g_j} k_i k_j\right)$$

$$+ \sum_i \left(\frac{A_{ii}}{2} \log\left[p_{g_i g_i}\right] - p_{g_i g_i} k_i k_i\right) + constants$$

$$= \frac{1}{2} \sum_{i,j} \left(A_{ij} \log\left[p_{g_i g_j}\right] - p_{g_i g_j} k_i k_j\right) + constants$$

Optimising the parameters

- We want to maximise this expression w.r.t. to p_{rs} and g_i.
  - First rewrite, where:
  - m_{rs} is the number of edges between group r and s.
  - κ_r is the sum of the degrees of the vertices in group r.

$$\frac{1}{2}\sum_{i,j}\left(A_{ij}\log\left[p_{g_i g_j}\right] - p_{g_i g_j}k_i k_j\right) = \frac{1}{2}\sum_{r,s}\left(m_{rs}\log[p_{rs}] - p_{rs}\kappa_r\kappa_s\right)$$

- Not surprisingly the p_{rs} attain the maximum at $p_{rs} = \frac{m_{rs}}{\kappa_r\kappa_s}$
  - Probabilities of edges between group i and j are (degree corrected) proportional to the actual number of edges.
  - Can be obtained formally through differentiation and looking for where the derivatives are zero.
- With prior knowledge: the p_{rs} can also be fixed beforehand.

Optimising the groups

- Final result: maximise this expression w.r.t. to the assignment of vertices to groups.
  - m_{rs} is the number of edges between group r and s.
  - κ_r is the sum of the degrees of the vertices in group r.

$$\frac{1}{2}\sum_{r,s}\left(m_{rs}\log\left[\frac{m_{rs}}{\kappa_r\kappa_s}\right]+m_{rs}\right) \qquad \frac{1}{2}\sum_{r,s}\left(m_{rs}\log\left[\frac{m_{rs}}{\kappa_r\kappa_s}\right]\right)$$

- Now we have an optimisation problem.
  - Can be solved by same algorithms as before.
  - E.g.: simple node moving. Or many others.
  - Works really well for graph that look like the statistical model.

Final Comments

- Statistical inference based methods allow to formally formulate a model of the data.
  - This gives the method a strong foundation.
  - Also works very well in practice.
- Disadvantage compared to Louvain is:
  - Number of communities need to be set before hand.
  - Louvain quickly finds this number of communities as well.
- Statistical method (under certain conditions):
  - If the statistical model of the graph is correct (assumption), and it cannot find the communities (assuming optimal optimisation), then they cannot be distinguished by any algorithm.

Universiteit Utrecht

[Faculty of Science
Information and Computing Sciences]

The experimental term paper

Network Science 2019/2020
Johan M. M. van Rooij
Erik Jan van Leeuwen

The experimental term paper

- Part of this course is writing a term paper
  - Either: the experimental term paper,
  - Or: the literature term paper.
- Term papers are peer-reviewed.
  - Feedback can be processed before submitting the final version.
- Experimental term paper:
  - Do an experimental evaluation of existing algorithms.
  - You can use existing implementations.
  - You should implement at least one algorithm yourself.
- Experimental projects are individual.
  - You should write your own code.
  - You are allowed to discuss experimental setup and analysis.

Focus?

- You can focus on one of two directions:
  - A comparative study of known algorithms.
  - Your own (modification to a known) algorithm and compare it to the original and/or other algorithms.
- Of course you can do both.
  - Note: it's better to do one thing right than two things half.

- Additionally, you need to focus on:
  - Setting up your experiments in a scientific manner.
  - Reporting on your experiments in a scientific manner.
- For guidelines, e.g., see Angriman et al. Guidelines for Experimental Algorithmics in Network Analysis.

Schedule and deadlines

- **Monday, April 27**: Submit topic selection.
  - E-mail Erik Jan your top-3 choices in CSV
    - [Student number], 1A, 3A, 2C.
  - Erik Jan will run a randomized assignment algorithm.
  - Results on Tuesday.
- **Monday, May 4**: Submit short proposal.
  - Formulate research question and approach.
  - One paragraph motivating your choices.
  - Feedback by me.
- **Tuesday, June 2**: High-quality draft for peer-review.
- **Monday, June 8**: Peer feedback using PeerGrade
- **Monday, June 15**: Final version! Use the feedback.

Possible topics

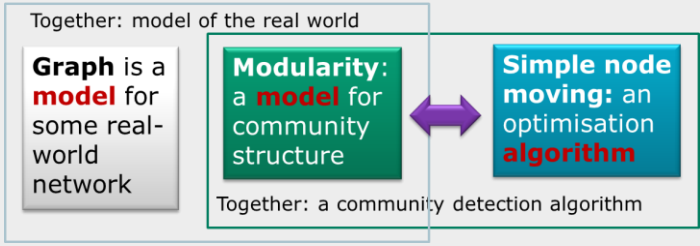
- We have selected five possible topics.
  - All topics are described at a high level.
  - You should formulate research questions that you want to answer yourself.
  - You can propose your own topic if you want.
- For inspiration, consider this (or any other) comparative study: Lancichinetti and Fortunato, Community detection algorithms: a comparative analysis.
  - (Partly) reproducing and verifying these results can be part of your experiments.
  - However, we do expect something of your own:
    - Testing some hypothesis of your own.
    - Testing different algorithms, your own algorithm, etc.

The experimental term paper - Network Science

POSSIBLE TOPICS

6. Compare different algorithms for modularity

- Modularity is a popular model for community structure.
  - Many different algorithms to optimise modularity exists.

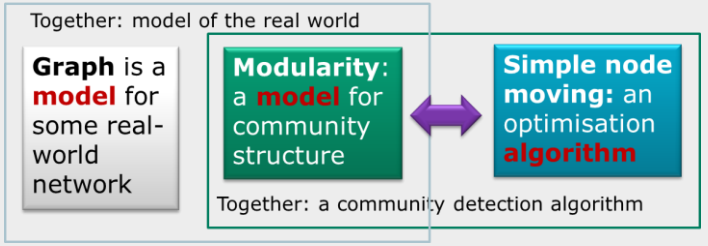


- Clause-Newma-Moore (NetworkX)
- Louvain (extension to NetworkX)
- Course's books have algorithms.
  - Greedy algorithm, simple node moving, spectral decomposition.
- Algorithms from optimisation literature:
  - Simulated annealing, tabu search, branch & bound.

- Possible questions:
  - Compare algorithms to (near) optimal modularity.
  - Compare scaling of algorithms in speed and quality.

7. Compare different measures of community structure

with the optimisation algorithm fixed



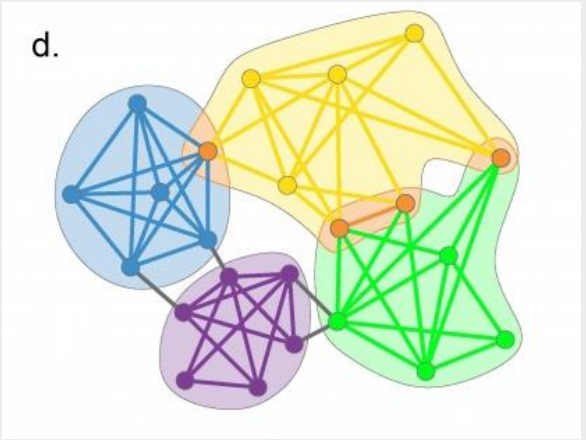
- Although the community model and the optimisation algorithm are often linked, some algorithms work with any community model.
  - Simple fast heuristics such as greedy or simple node moving.
  - Simple agglomerative heuristics, e.g., the first phase of Louvain.
- Models:
  - Modularity, MapEquation (from the InfoMap method), statistical models, or others (e.g., see topic document).
- Comparing speed probably not very useful unless you compare models for which the same optimisations exits.
  - Comparing different models to a ground truth.
  - Compare quality of models (with near-perfect optimisation)
  - Or find out what models (that allow fast evaluation of deltas) remain useful with a very fast optimisation heuristic.

8. Fast heuristics on large graphs

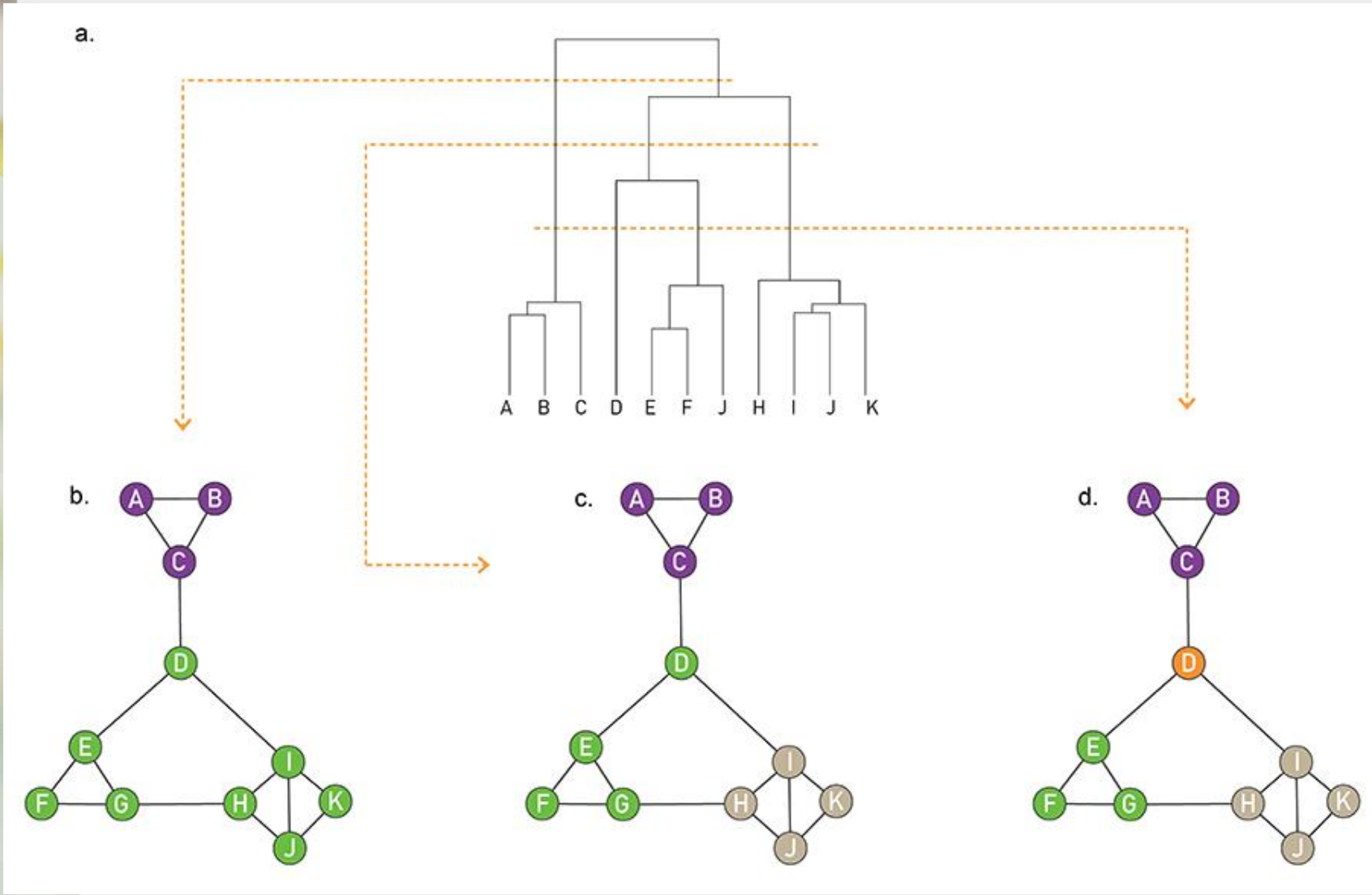
- We use fast heuristics, because graph can be (very) large.
  - Then only very fast algorithms are practical.
- Compare different existing (or your own) algorithms and investigate how they scale to larger and larger graphs.
  - Which algorithms remain useful?
  - Does the output quality significantly degrade at some point?
  - How does the runtime scale for really large graphs.
  - Can you devise an algorithm of your own that scales really well?
- We recommend using (different) graph generators to generate comparable graphs of different sizes.
  - Also, think about the setup of (long running) experiments.

9. Finding overlapping communities

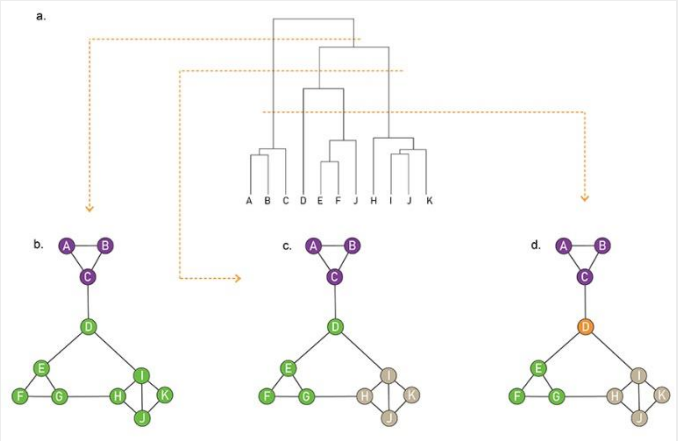
- Standard community detections focusses on partitioning vertices into communities – this is often not realistic.
  - Many algorithms for overlapping communities exist also.
  - Algorithms were a vertex may be in multiple communities.
  - And algorithms that cluster edges not vertices.
  - See: CFinder a.k.a. clique percolation, statistical methods, edge clustering, etc.
- We can ask many of the same questions?
  - Compare quality of algorithms?
  - Compare speed of algorithms?
  - Which remain useful for large graphs?
  - What are good models of communities?



10. Comparing hierarchical dendrogram-based methods



5. Comparing hierarchical dendrogram-based methods



- Some networks have a natural hierarchical structure.
  - Road networks.
  - Many examples in biology.
- This is slightly more difficult.
- You should think about:
  - What is the ground truth and how to compare against it.
  - Note that you can compare a community partition to a dendrogram.
- Compare different strategies:
  - Different top-down methods for bisection into 2 communities.
  - Different bottom-up methods to merge smaller communities.
  - Or methods to start with a dendrogram and then determine the best cut into communities.

The experimental term paper - Network Science

SOME GUIDELINES ON EXPERIMENTS

Lets be clear on this…

- We do not want you to just:
  - Implement some algorithms
  - Run them on many graphs
  - Collect results in tables and graphs
  - Conclude that algorithm A is faster/better than algorithm B in some cases, and C is faster in others, etc.
- Can we truly justify any claim done?
  - Can you conclude anything on the scaling behaviour?
  - Is the difference due to the algorithm or the implementation?
  - Or maybe the class of graphs used fit one better than the other?
  - Is the difference statistically significant?
- The first bullet can be an exploratory experiment done before formulating an actual research question/hypothesis.

Formulating hypotheses or research questions?

- We want you to formulate one or more hypotheses or research questions that you investigate using experiments.
  - Possibly after an exploratory first experiment.
  - Note that to test a hypothesis you always need to do new experiments.
  - Given enough data there always exist hypotheses consistent with the data.
- Hypothesis or research questions about:
  - Speed or quality of an algorithm.
  - Speed or quality of an algorithm relative to another algorithm.
  - How parameters of/variations of the input affect an algorithm.
  - How parameters of the algorithm effect its behaviour.

Hypothesis or research question

- We often take about a formulating a hypothesis before doing an experiment.
  - Hypothesis can sometimes be a confusing term here.
- A good way to look at the scaling of the runtime an algorithm on a class of generated graphs is:
  - Run the algorithm on many graphs of different sizes.
  - Fit a statistical model to its running time.
  - E.g.: fit running time as follows, with ε normally distributed.

$$T(n) = a \cdot n^b \cdot \varepsilon$$

  - Estimate the parameters of the model.
  - Consider confidence intervals or significance levels.

Gather appropriate data

- Gather data that is relevant for your hypotheses or research questions.
  - Real world graph or graph generators?
  - Do we need a ground truth?
  - Do we need graphs with/without certain properties?
- How much data do we need?
  - Uncertainty in many statistics of the results often scales with $\sqrt{\frac{s^2}{n}}$, where s the sample variance and n the number of data points.

Graph generators

- Advantages:
  - Have ground truth.
  - Can produce graphs of any size.
- Disadvantages:
  - Choice of parameters can greatly influence results.
  - Are not always a model for real-world instances.
  - Can produce graphs with (in expectation or in high probability) inherent properties that are beneficial to some algorithm, while disadvantageous to other algorithms.

What to compare?

- Which algorithms?
  - Do we want to compare to state-of-the-art?
- What algorithm parameters do we choose?

- What is the influence of:
  - The hardware?
  - The specific software implementation?
- What do we measure?
  - CPU time of Wall-clock time?

- Do we repeat runs with the same data?
  - Effects of randomisation (different seed)
  - Average out random due to operating system?

Repeatability

- A good experiment is reproducible.
  - For example by a peer-reviewer – we expect this!
- This need not be difficult.
  - Use scripts or notebooks.
  - Use a language / an environment that is common to others.
  - Put your own code on GitHub.
- Doing this for larger experiments can be a lot of work.
  - Version control.
  - Fixing versions of all dependencies.
  - Build scripts besides scripts for the experiment.
  - Storing output files in a standard format so that the results can be re-analysed without repeating all experiments.

Analysing the results

- If you want to justify claims based on your experimental results… use statistics!
  - Hypothesis testing? Parameter estimation?
- I'm not going to give a statistics lecture here.
  - See the research methods course in the bachelor.
  - Or many other good resources…
- Also, be careful to not only report statistics of the results.
  - Many different distributions of numbers can have the same mean and variance.
  - Use graphs for important measurements!
- Finally, good statistical analysis is based on assumptions; your experiment is often also based on assumptions…
  - Do not forget to report on these!

Thats all folks… go experiment