KU LEUVEN

Faculty of Engineering Science
Master in Artificial Intelligence
ECS Option
Academic year 2019-2020

ALSSN
Analysis of Large Scale Social Networks

B-KUL-H0T26A

By Bart Thijs

Course Notes
Created by Elina Oikonomou

Three Criteria for Measurement and analysis in Projects

1. Validity: the answer corresponds to the question, the extent to which a conclusion or measure accurately represents the real world:
   - External validity is related to the generalization of the results of your analysis of a particular data-set. Does the results hold in other situations, with different data, with different people?
   - Construct validity is related to the claim that a certain property really measures what it is supposed to measure.
2. Reliability: overall consistency of the analysis, degree to which results and conclusions remain consistent after applying different methodologies eg. different clustering techniques
3. Scalability: extend of proposed methodologies' ability to cope with large scale networks, taking into account time and space complexity
   - Time Complexity: This refers to the time needed for the analysis as a function of the input size of the network
   - Space Complexity: This refers to the storage both in memory or in disk needed for storing and processing the data. In a second order it also refers to the IO-operations and network traffic required for data processing

During exam, we will be asked on these topics, eg. scalability of a methodology

1. Centrality measures: most central node
   - Degree: number of links associated with a node
   - Betweenness: times that a node acts as a bridge between pairs of other nodes

2. Observed and expected relation between these two

Degree of nodes, not distributed in a normalized way, there are many nodes with low degrees and few nodes with high degrees. This is also observed in social networks: there are lots of accounts with few followers/friends and few accounts with lots of followers
For betweenness, the observation is a truncated power-law behavior

Expected relationship: those 2 should be similar

3. Explanation: communities

Definition of community: high modularity

Hubs: used to connect one community to other communities eg. NYC

4. What about validity, reliability and scalability of the data source, measures and explanation?

Taking degree as a proxy of betweenness is not the most reliable method

We can't say if the analysis is useful because we didn't define the purpose
For example if the purpose is to optimize the environmental footprint of air travel or prevent the spread of disease, then taking measures for Paris makes sense, but not for Anchorage (Alaska). The data that we miss is the population of the city, or number of number of passengers. When we look at the weighted network and we look at degree (no of edges), then we get a different view of degree. So we can have 2 completely different cases of high degree: few connections with lots of passengers and many connections with few passengers each.
Betweenness: shortest path, when we look at weighted path we can indicate a kind of similarity / distance measure. We can define the cost of using a connection, similarity: the higher the value the higher the probability that we will take this connection. We can then take the cost of the complete path. Shortest path has an influence on how we calculate betweenness.

Extra topic: Graph embeddings (SOTA)

Lecture 1: Basic Network Concepts

Karate Club network: 34 members and interactions of members outside the club
The club falls apart in 2 distinct groups
Members: nodes, vertices
Interactions: edges
Clustering algorithm
Unidirectional network, edges don't have direction
Directed edges: arks
Nodes and edges can have attributes:
- Nodes can have a name or a size
- Edges can have a weight: weighted network

Formal definition: a graph is an ordered pair of 2 distinct sets
G = (V,E)
V = set of vertices/nodes
E = set of edges which are pairs of elements from V

Network analysis focuses on properties at two distinct levels:
Local Level: this discusses the role or position of individual nodes or small subgraphs within the network. Eg. the role of node 20 in the karate club: he has a limited number of links but has contacts with both the trainer and the administrator → central role of passing information from one group to the other
Global Level: this type of analysis focusses on the large scale structure of the network or large subgraphs

- Largest connected component: is the component containing most vertices within the network
- Singleton: a single node which has no connection to any of the other nodes in the network

Matrix representation of a network: adjacency matrix, row i, column j
A: N by N
A_{ij} = 1 if link from j to I, otherwise 0

In undirected network:
$A_{ij} = A_{ji}$

$$k_i = \sum_{j=1}^{N} A_{ji} = \sum_{j=1}^{N} A_{ij}$$

In directed network:
$k_i^{in} = \sum_{j=1}^{N} A_{ij}$, $k_i^{out} = \sum_{j=1}^{N} A_{ji}$

The degree can be calculated by taking the sum of the row or a column.
In-degree: sum over the rows
Out-degree: sum over the columns
All elements on the diagonal are all 0s, unless loops are allowed: a node it's connected to itself

Weight adjacency matrix: in a weighted network, the links can have a value: $A_{ij} = w_{ij}$
Values can represent:
- Similarities: higher weight → stronger tie between nodes.
  Weighted degree: $k_i^{w} = \sum_{j=1}^{N} w_{ij}$
- Distances: higher weight → nodes are further away from each other, a weaker tie

The strength of the connection can be represented by the boldness of the link, strongly connected nodes have a position more aligned to each other. Coloring and labels: higher level grouping/clustering of the nodes

<u>Paper: The worldwide air transportation network: Anomalous centrality, community structure, and cities' global roles</u>

1. Which centrality measures were used?

The 2 centrality measures used initially are:
Degree: no of direct links with other cities
Betweenness: of city i is defined as the number of shortest paths connecting any two cities that involve a transfer at city i
Normalized Betweenness: bi = Bi/<B>, where <B> represents the average betweenness for the network

2. What is the observed and expected relation between these measures?

There is an assumption that "most connected cities are the most central". However in this case there are nodes with small degree and large centrality, which considered as anomalies. This kind of behavior is not present in other network cases, such as the Internet.

3. What explanation is given?

These anomalous centralities derive from the existence of local communities and cities with low degree (low number of connections with other cities), but high betweenness. There is need to identify the communities in the air transportation network and, moreover, establish new ways to characterize the role of each city based on its pattern of intracommunity and intercommunity connections and not merely on its degree.

4. What about validity, reliability and scalability of the data source, measures and explanation?

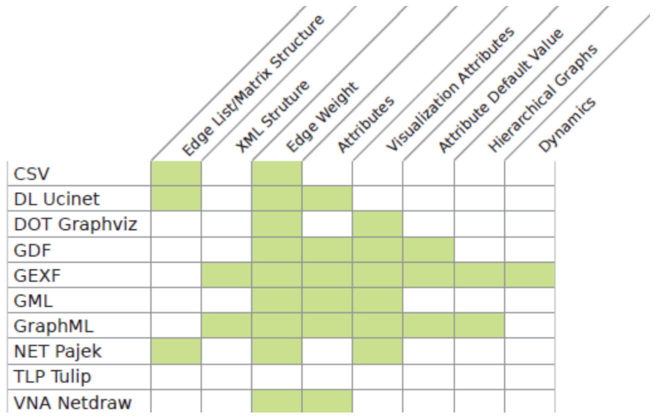
The validity, reliability and scalability of this paper is highly challenged, particularly on the below point:
- To define degree and betweenness of the node, a binary analysis has been done. In case weights are assigned to the connections between the cities, indicating the number of passengers, no of flights or actual distance between the cities, the rankings of degree and betweenness would be different vs what is presented on the paper.
- As a consequence, we cannot define if the analysis is useful, as the objective or purpose is not defined. For instance, if we want to optimize the environmental footprint of air travel or prevent the spread of disease, then taking measures for Paris makes sense, but not for Anchorage (Alaska).
- Regarding the Database, OAG is an authoritative database with worldwide routes, so that minimizes the sample selection biases. However, there is missing information useful for the analysis, such as no of passengers etc.

Lecture 2: Storing and Processing Graph Data

Graph (abstract data type)
From Wikipedia, the free encyclopedia

Several data formats have been developed for storing network data. Some of them are developed independently from graph processing software, others are specific to certain applications. Most software, however, can deal with many formats.

File format: XML, JSON, Plain Text, Binary format, Serialized Objects



	Edge List/Matrix Structure	XML Struture	Edge Weight	Attributes	Visualization Attributes	Attribute Default Value	Hierarchical Graphs	Dynamics
CSV	🟩		🟩					
DL Ucinet	🟩		🟩	🟩				
DOT Graphviz			🟩		🟩			
GDF			🟩	🟩	🟩	🟩		
GEXF		🟩	🟩	🟩	🟩	🟩	🟩	🟩
GML			🟩	🟩	🟩			
GraphML		🟩	🟩	🟩	🟩	🟩	🟩	
NET Pajek	🟩		🟩		🟩			
TLP Tulip								
VNA Netdraw			🟩	🟩				

Task 1:
Search for graph data formats and score each of them on the above-mentioned criteria. Try to find the most common graph processing software and indicate which formats are supported by these programs. Use the discussion forum for your results

	CSV	GraphViz DOT	GDF	GML	GraphML	GEXF
Representation	Matrix	Matrix	Matrix	Lists	Lists	List
Format	Plain text	Plain Text/XML	Plain text	Plain text	XML	XML
Undirected/Directed	yes	yes	Yes	Yes	Yes	Yes
Unweighted/weighted	yes		Yes		Yes	Yes
Node Labels		Yes	Yes	Yes	Yes	Yes
Edge Labels		Yes	Yes	Yes	yes	No
Multiple node edges	yes		No			Yes
Layout/Visualization		Yes	Yes		Yes	Yes
Dynamic Graphs						Yes

1. CSV Format

A comma-separated values (CSV) file is a delimited text file that uses a comma to separate values. A CSV file stores tabular data (numbers and text) in plain text. Each line of the file is a data record. Each record consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format.
The term "CSV" also denotes some closely related delimiter-separated formats that use different field delimiters, for example, semicolons. These include tab- separated values and space-separated values. A delimiter that is not present in the field data (such as tab) keeps the format parsing simple. These alternate delimiter- separated files are often even given a .csv extension despite the use of a non- comma field separator. This loose terminology can cause problems in data exchange. Many applications that accept CSV files have options to select the delimiter character and the quotation character.

2. GraphViz DOT Format

DOT is the text file format of the suite GraphViz, an open source graph visualization software. It has a human-readable syntax that describes network data, including subgraphs and elements appearances (i.e. color, width, label).

3. GDF Format

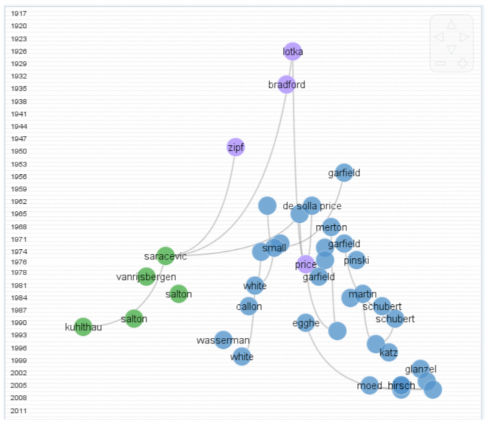
GDF is the file format used by GUESS. It is built like a database table or a coma separated file (CSV). It supports attributes to both nodes and edges. A standard file is divided in two sections, one for nodes and one for edges. Each section has a header line, which basically is the column title. Each element (i.e. node or edge) is on a line and values are separated by coma. The GDF format is therefore very easy to read and can be easily converted from CSV.

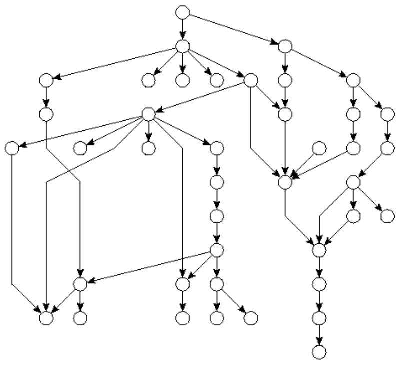
4. GML Format

The Graph Modeling Language (GML) is a file format for graphs designed to represent arbitrary data structures and characterized by portability, simple syntax, extensibility and flexibility. A GML file consists of hierarchical key-value lists. Even if GML was bound to a specific system, namely Graphlet., then it has been overtaken and adopted by several other drawing graphs systems, such as Pajek, yEd, LEDA and NetworkX

5. GraphML Format

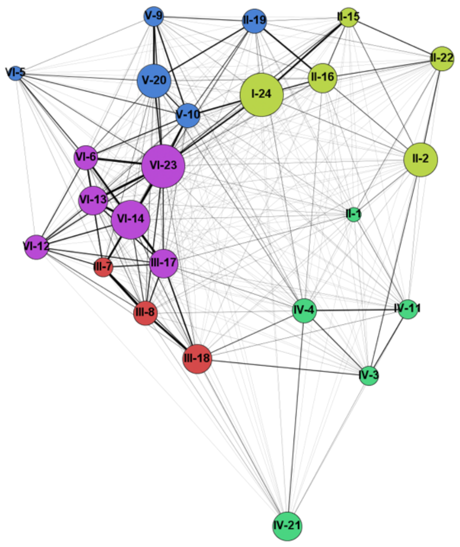
The GraphML file format uses .graphml extension and is XML structured. It supports attributes for nodes and edges, hierarchical graphs and benefits from a flexible architecture. Gephi supports a limited set of this format (no sub-graphs and hyperedges). This format is supported by NodeXL, Sonivis, GUESS and NetworkX.





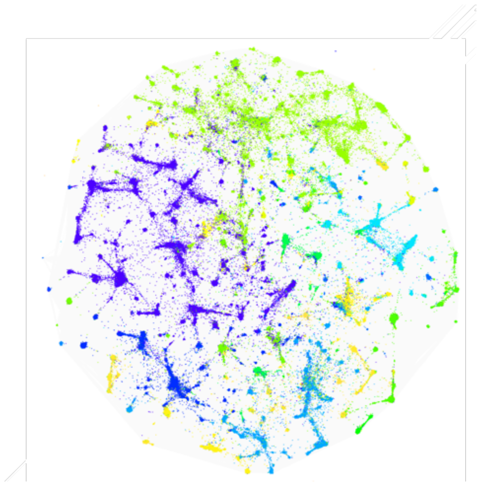
Spring Based Models
Edges are associated with attractive power. Absence of edge can result in repulsive power. Visualization results in a system with highest equilibrium.

- Kamada Kawai (1989) only uses attractive power
- Fruchterman-Rheingold (1991) uses sum of forces to decide the direction to move with constant step width



Spring Based Models
Extension of force-directed layouts have been proposed and implemented

- Force Atlas & Forse Atlas2
- Yifan Hu
- OpenOrd or DrL



Dimensionality Reduction: MDS
High dimensional distance data is reduced to a 2-dimensional space in such a way that the original distance between nodes remain preserved

From Wikipedia, the free encyclopedia, https://en.wikipedia.org/wiki/Multidimensional_scaling

Multidimensional scaling (**MDS**) is a means of visualizing the level of similarity of individual cases of a dataset. MDS is used to translate "information about the pairwise 'distances' among a set of n objects or individuals" into a configuration of n points mapped into an abstract Cartesian space.
More technically, MDS refers to a set of related ordination techniques used in information visualization, in particular to display the information contained in a distance matrix. It is a form of non-linear dimensionality reduction.
Given a distance matrix with the distances between each pair of objects in a set, and a chosen number of dimensions, *N*, an MDS algorithm places each object into *N*-dimensional space such that the between-object distances are preserved as well as possible. For *N=1, 2,* and *3*, the resulting points can be visualized on a scatter plot.

Types
- Classical multidimensional scaling
- Metric multidimensional scaling (mMDS)
- Non-metric multidimensional scaling (nMDS)
- Generalized multidimensional scaling (GMD)

Task 3:
Investigate existing graph processing software and make an inventory of implemented graph visualization algorithms. Discuss these techniques and try to identify the criteria that were used by the authors to obtain a 'good' visualization. Evaluate them with respect to Scalability

Graph Processing Software: https://www.3pillarglobal.com/insights/a-quick-look-into-the-popular-graph-databases

NEO4J
Neo4j is the market leader in GraphDB. It is highly performant, scalable, and flexible. It is equipped with a rich UI, called the Neo4j browser, that is used for queries, visualization, and data interaction. It has compatibility with most programming platforms like Java, Nodejs, PHP, Python, .NET, etc. It supports the Cypher query language, whose syntax naturally depicts data and relationships. It has a good integration support with Apache Spark, Elasticsearch, MongoDB, Cassandra, and Docker. The community edition is free under GPL v3 license.

ORIENT DB
Orient DB is a high speed, scalable, and reliable GraphDB. It has a high and constant traversing speed that is not affected by database size. User, role, and record level security is built-in. It has a good compatibility with most programming platforms like Java, Nodejs, PHP, Python, .NET, etc. It has a strong community support and is licensed under Apache 2.

TITAN DB
Titan DB is highly scalable and can store and query graphs containing hundreds of billions of vertices and edges distributed across a multi-machine cluster. Titan DB is transactional and supports various backend stores like Cassandra, HBase, and BerkeleyDB. It supports integration with Elastic Search, Lucene, and Solr, and also supports the Gremlin query language. It is licensed under Apache 2.

APACHE GIRAPH
It is an Iterative Graph processing framework backed by Apache Hadoop. Giraph jobs are highly scalable as they run on Hadoop Cluster. Giraph can greatly increase the efficiency of graph computation on a huge dataset. It is licensed under Apache 2. We wanted to take a deep dive into these graph databases to understand how they stacked against each other. At the outset, we eliminated TitanDB because a stable 1.0 version had not been released at the time. We also skipped Apache Giraph because it is backed up only by Hadoop. This left Neo4j and OrientDB for more exploration.

Paper: OpenOrd: An Open-Source Toolbox for Large Graph Layout (DrL), Martin et al.

Suppose we have an undirected weighted graph *G* = (*V*,*E*), where the vertices are given by *V* = {*v*1 , . . . , *vn* } and the edges are given by *E* = {e_{ij} }. Let *W* = (w_{ij}) be the adjacency matrix corresponding to the graph *G* so that edge e_{ij} has weight w_{ij}. Since the graph is undirected, we know that $w_{ij} = w_{ji}$ so that *W* is symmetric.
The goal of OpenOrd is to draw *G* in two dimensions. Let x*i* = (*xi*,1 , *xi*,2) denote the position of *vi* in the plane. OpenOrd draws *G* by attempting to solve

$$\min_{x_1,\dots,x_n} \sum_i \left(\sum_j (w_{ij} d(x_i, x_j)^2) + D_{x_i} \right)$$

where *Dxi* denotes the density of the points x1,...,x*n* near x*i*. The sum in (1) contains both an attractive and a repulsive term. The attractive term ∑*j*(*wijd*(x*i*,x*j*)2)) attempts to draw together vertices which have strong relations via *wij*. The repulsive term *Dxi* attempts to push vertices into areas of the plane that are sparsely populated.
Greedy optimization procedure: update the position of each vertex by optimizing the inner sum $\sum_j (w_{ij} d(x_i, x_j)^2) + D_{x_i}$
All vertices are initially placed at the origin, and the update is repeated for each vertex in the graph to complete one iteration of the optimization. The iterations are controlled via a simulated annealing type schedule which consists of five different phases: liquid, expansion, cool-down, crunch, and simmer.
Hyperparameters: temperature, attraction, and damping: control how far vertices are allowed to move.
At each step of the algorithm, we compute two possible vertex moves.
- The first possible move is always a random jump, whose distance is determined by the temperature.
- The second possible move is analytically calculated (known as a barrier jump22). This move is computed as the weighted centroid of the neighbors of the vertex.

2.2) Parallelization How: using a coarsening procedure that merges adjacent nodes at random and adds their weights. This uses average link-clustering. This coarsening procedure also helps to improve the deficiencies of the edge-cutting step.
3) criteria:
- visually appealing layout without overlapping of clusters (use attraction and repulsing to find appropriate layout)
- good performance for large networks.

Paper: A Comparison of Two Techniques for Bibliometric Mapping: Multidimensional Scaling and VOS

Multidimensional Scaling – Similarity measures

Direct / local similarity measures: determine the similarity between two items by applying a normalization to the co-occurrence frequency of the items. The underlying idea is that co-occurrence frequencies can be interpreted as similarities only after one has corrected for the fact that for some items the total number of occurrences or co-occurrences may be much larger than for other items.

Indirect / global similarity measures: determine the similarity between two items by comparing two vectors of co-occurrence frequencies. This is based on the idea that the similarity of two items should depend on the way in which each of the two items is related to all other items. The more two items have similar relations with other items, the more the two items should be considered similar.

Association strength / proximity index / probabilistic affinity index: $AS_{ij} = \frac{c_{ij}}{c_i c_j}$

Cosine similarity:
$$\mathrm{COS}_{ij} = \frac{\sum_{k \neq i,j} c_{ik} c_{jk}}{\sqrt{\sum_{k \neq i,j} c_{ik}^2 \sum_{k \neq i,j} c_{jk}^2}}.$$

Technique on Multidimensional Scaling
Aim of MDS: to locate items in a low-dimensional space in such a way that the distance between any two items reflects the similarity or relatedness of the items as accurately as possible. The stronger the relation between two items, the smaller the distance between the items.
s_{ij}: similarity between items *i* and *j* given by some direct or indirect similarity measure
p_{ij} proximity (i.e., a similarity or dissimilarity)
weight wij (wij ≥ 0). Typically $w_{ij} = 1$ for all *i* and *j*

Stress function:
$$\sigma(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \frac{\sum_{i<j} w_{ij} (f(p_{ij}) - \|\mathbf{x}_i - \mathbf{x}_j\|)^2}{\sum_{i<j} w_{ij} f(p_{ij})^2}.$$
f: transformation function of proximities p_{ij}
in 2D: Euclidean distance measure

MDS determines the locations of items in a map by minimizing the (weighted) sum of the squared differences between on the one hand the transformed proximities of items and on the other hand the distances between items in the map.
The transformation function *f* has to be increasing when the proximities p_{ij} are dissimilarities and decreasing when the proximities p_{ij} are similarities.

Ratio MDS: treats the proximities pij as measurements on a ratio scale, f is a linear function without an intercept. We can't use this when proximities pij are similarities
Interval MDS: treats the proximities pij as measurements on an interval scale, f can be any linear function.
Ordinal MDS: treats the proximities pij as measurements on an ordinal scale, f can be any monotone function.

Most typically: The proximities pij are typically set equal to the similarities sij, which means that ratio MDS cannot be used.
The stress function in Equation 3 can be minimized using an iterative algorithm.

VOS
VOS aims to locate items in a low-dimensional space in such a way that the distance between any two items reflects the similarity or relatedness of the items as accurately as possible.

VOS requires as input a similarity sij (sij ≥ 0). VOS treats the similarities sij as measurements on a ratio scale. The similarities sij are typically calculated using the association strength. VOS determines the locations of items in a map by minimizing
$$V(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \sum_{i<j} s_{ij} \|\mathbf{x}_i - \mathbf{x}_j\|^2$$
Subject to:
$$\frac{2}{n(n-1)} \sum_{i<j} \|\mathbf{x}_i - \mathbf{x}_j\| = 1.$$
The idea of VOS is to minimize a weighted sum of the squared distances between all pairs of items. The squared distance between a pair of items is weighted by the similarity between the items. To avoid trivial solutions in which all items have the same location, the constraint is imposed that the average distance between two items must be equal to one.

Ideal location of item i:
$$\mathbf{x}_i^* = \frac{\sum_{j \neq i} s_{ij} \mathbf{x}_j}{\sum_{j \neq i} s_{ij}}.$$

Items will in general not be located exactly at their ideal location. However, due to the objective function, items at least tend to be located close to their ideal location.

Time Complexity: O(V^3) → very expensive

Comparison of Dijkstra’s and Floyd–Warshall algorithms

- Dijkstra’s Algorithm is one example of a single-source shortest or SSSP algorithm, i.e., given a source vertex it finds shortest path from source to all other vertices.
- Floyd Warshall Algorithm is an example of all-pairs shortest path algorithm, meaning it computes the shortest path between all pair of nodes.

Time Complexities :

- Time Complexity of Dijkstra’s Algorithm: O(E log V)
- Time Complexity of Floyd Warshall: O(V^3)

Other Points:

- We can use Dijsksta’s shortest path algorithm for finding all pair shortest paths by running it for every vertex. But time complexity of this would be O(VE Log V) which can go (V^3 Log V) in worst case.
- Another important differentiating factor between the algorithms is their working towards distributed systems. Unlike Dijkstra’s algorithm, Floyd Warshall can be implemented in a distributed system, making it suitable for data structures such as Graph of Graphs (Used in Maps).
- Lastly Floyd Warshall works for negative edge but no negative cycle, whereas Dijkstra’s algorithm don’t work for negative edges.

Random walk
A random walk in a graph starting from the node and of length k is a sequence of nodes and edges starting with node i and randomly chosen subsequent connecting edges and nodes. These walks can be considered as Markov Chains, each step is independent from the previous one
Probability of step from node i to j at time t in a random walk (X0, X1, X2, …, Xk, …) is equal to:

$$p(X_k = j) \;=\; p(X_{k-1} = i)A_{ij}\frac{1}{d_i}$$

Laplacian matrix
Suppose D is diagonal degree matrix and A is adjacency matrix then the Laplacian matrix L is defined as: L = D – A
For an undirected and unweighted network where:

$$L_{ij} = \begin{cases} \deg(v_i) & \text{if } i=j \\ -1 & \text{if } i{=}/j \text{ and } v_i \text{ and } v_j \text{ are adjacent} \\ 0 & \text{otherwise} \end{cases}$$

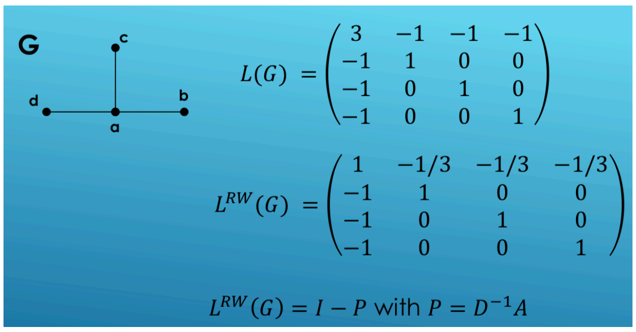
Properties:

- First eigenvalue is 0 with eigenvector $v_0 = (1)^n$
- All eigenvalues are nonnegative (0,1,1,4)
- The multiplicity of 0 is the number of connected components
- The second smallest value is the Algebraic Connectivity

Connectivity: minimum number of nodes/edges to be removed to break connection graph

Suppose D^{-1} is diagonal matrix with inverse of degree, then $L^{RW} = D^{-1} L$, with

$$L^{RW}(G) = \begin{cases} 1 & \text{if } i=j \text{ and } \deg(i) > 0 \\ -1 / \deg(i) & \text{if } i{=}/j \text{ and } v_i \text{ and } v_j \text{ are adjacent} \\ 0 & \text{otherwise} \end{cases}$$



$$L(G) \;=\; \begin{pmatrix} 3 & -1 & -1 & -1 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{pmatrix}$$

$$L^{RW}(G) \;=\; \begin{pmatrix} 1 & -1/3 & -1/3 & -1/3 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{pmatrix}$$

$L^{RW}(G) = I - P$ with $P = D^{-1}A$

→ Laplacian: https://www.youtube.com/watch?v=EjpMnU79neo

Markov Transition matrix
Matrix that holds the probabilities of moving from one note (state) to the next. Applicable on directed, weighted graph, with:

$$P_{ij} = \begin{cases} w_{ij} / \text{outdeg}(i) \text{ if } i{=}/j \text{ and directed edge from } i \text{ to } j \\ 0 \end{cases}$$

Traversals and centrality
Centrality: refers to the importance of a node in a network. This importance can be expressed based on. Its role in the information flow

Betweenness measure for a vertex i that is equal to the ***net*** number of times that the message passes through i on its journey, averaged over a large number of trials of the random walk. The full random-walk betweenness of vertex i will then be this value averaged over all possible source/target pairs s, t.

Example: disease spread, random-walk model represents better this case, people don't use a "shortest" path towards a target victim
Actually most of real world scenarios work with the random walk scenario: eg. the spread of a video or a meme at social media, it's random, there is no shortest path that is used to spread through the users.

Paper: Multi-agent random walks for local clustering on graphs,

Purpose: Discovering Local communities

- Random walks are performed by multiple walkers (agents) pairwise connected by a 'rope' of fixed length
- The probability that connected walkers travel over a bridge/bottleneck from one cluster to another is small
- If transition probability is p from random walk, then the probability becomes p^a for RW with a agents

Local graph clustering, random walk at the point of interest until stopping criteria is met
Suggestion: multi-agent random walk, several agents connected by fixed rope of length l
→ less tendency to mistakenly merge two different clusters than the original random walk

In large networks, global clustering is infeasible
Local clustering: find cluster if a particular vertex of interest, eg. community of a particular person
But, cannot guarantee global clustering

Popular local clustering: Nibble Algorithm
Given the vertex v of interest, one considers a random walk (RW) on the graph that starts at vertex v. This random walk is biased in the sense that it only jumps along "important" edges, unimportant edges (with low weight) are ignored. The random walk runs until a certain stopping criterion is met, for example when the conductance of the set of visited vertices gets too large. The set of visited vertices is then considered the community of v.

Benefits of random walk approach
- they locally explore the neighborhood of a vertex and with reasonably high probability stay inside a cluster rather than transitioning to a different cluster.
- they can be implemented with low space complexity

Benefits of Multi-agent random walk (MARW)
- probability of transitioning between two different clusters is significantly reduced in the MARW compared to the standard RW.
- the MARW discovers tighter clusters than the standard RW.

Spider walk: construct an alternative to the standard random walk that travels faster between different clusters. To this end, the authors run several random walks completely independently from each other. In this way one can speed up the global exploration of large graphs.

- agents move at the same time and are allowed to sit on the same vertex.
- we start with a agents in locations v1 , ..., va .
- Step: a independent simple random walks, one for each agent.
- check if the new vertices still satisfy the condition on the rope length.
    - If yes, we accept the step, otherwise we discard it.
    - Discarding many suggested new states is not a problem in practice if the out-degree of each vertex is not considerably big.
    - For large degree vertices: consider variant where agents do not move all together, but one after the other, less positions get rejected

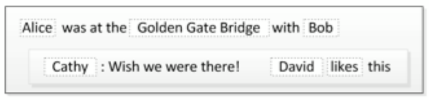
2 Parameters:
- number of agents: a
- small rope length: l

Probability of transitioning between different clusters becomes smaller

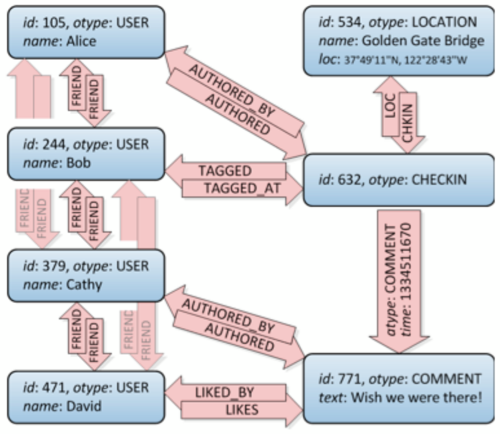
MARW-graph is the original graph with edge weights taken to the power a
→ whenever there is a choice of going in the direction of high or low edge weights, the MARW will choose the high weight version even more often than the standard RW
→ Consequently, it tends to move away from the boundary towards the center of a cluster.

Bounding the spectral gap
Goal: increase the probability of staying within a community.
One way to treat this formally is to analyze the spectral gap of the graph, in particular the second eigenvalue λ2 of the transition matrix.
It measures:
how clustered the graph is (for example, it can be used to bound the Cheeger cut) and can also be used to bound the mixing time of the random walk. The smaller λ2, the more clustered the graph is, and the larger the mixing time is.

Paper: Spiders in Random Environments, Gallesco, Muller, Popov


Alice was at the Golden Gate Bridge with Bob

Cathy : Wish we were there! David likes this



This simple example shows a subgraph of objects and associations that is created in TAO after Alice checks in at the Golden Gate Bridge and tags Bob there, while Cathy comments on the check-in and David likes it. Every data item, such as a user, check-in, or comment, is represented by a typed object containing a dictionary of named fields. Relationships between objects, such as "liked by" or "friend of," are represented by typed edges (associations) grouped in association lists by their origin. Multiple associations may connect the same pair of objects as long as the types of all those associations are distinct. Together objects and associations form a labeled directed multigraph.

For every association type a so-called inverse type can be specified. Whenever an edge of the direct type is created or deleted between objects with unique IDs id1 and id2, TAO will automatically create or delete an edge of the corresponding inverse type in the opposite direction (id2 to id1). The intent is to help the application programmer maintain referential integrity for relationships that are naturally mutual, like friendship, or where support for graph traversal in both directions is performance critical, as for example in "likes" and "liked by."

The set of operations on objects is of the fairly common create / set-fields / get / delete variety. All objects of a given type have the same set of fields. New fields can be registered for an object type at any time and existing fields can be marked deprecated by editing that type's schema. In most cases product engineers can change the schemas of their types without any operational work.

Associations are created and deleted as individual edges. If the association type has an inverse type defined, an inverse edge is created automatically. The API helps the data store exploit the creation-time locality of workload by requiring every association to have a special time attribute that is commonly used to represent the creation time of association. TAO uses the association time value to optimize the working set in cache and to improve hit rate.

There are three main classes of read operations on associations:

- Point queries look up specific associations identified by their (id1, type, id2) triplets. Most often they are used to check if two objects are connected by an association or not, or to fetch data for an association.
- Range queries find outgoing associations given an (id1, type) pair. Associations are ordered by time, so these queries are commonly used to answer questions like "What are the 50 most recent comments on this piece of content?" Cursor-based iteration is provided as well.
- Count queries give the total number of outgoing associations for an (id1, type) pair. TAO optionally keeps track of counts as association lists grow and shrink, and can report them in constant time

We have kept the TAO API simple on purpose. For instance, it does not offer any operations for complex traversals or pattern matching on the graph. Executing such queries while responding to a user request is almost always a suboptimal design decision. TAO does not offer a server-side set intersection primitive. Instead we provide a client library function. The lack of clustering in the data set virtually guarantees that having the client orchestrate the intersection through a sequence of simple point and range queries on associations will require about the same amount of network bandwidth and processing power as doing such intersections entirely on the server side. The simplicity of TAO API helps product engineers find an optimal division of labor between application servers, data store servers, and the network connecting them.

Implementation

The TAO service runs across a collection of server clusters geographically distributed and organized logically as a tree. Separate clusters are used for storing objects and associations persistently, and for caching them in RAM and Flash memory. This separation allows us to scale different types of clusters independently and to make efficient use of the server hardware.

Client requests are always sent to caching clusters running TAO servers. In addition to satisfying most read requests from a write-through cache, TAO servers orchestrate the execution of writes and maintain cache consistency among all TAO clusters. We continue to use MySQL to manage persistent storage for TAO objects and associations.

The data set managed by TAO is partitioned into hundreds of thousands of shards. All objects and associations in the same shard are stored persistently in the same MySQL database, and are cached on the same set of servers in each caching cluster. Individual objects and

Graph database models should at least address these tasks:

- Structuring, description and maintenance of data
- Retrieval or querying data, in line with data structure / graph data model

Three components:

1. Set of data structure types
2. Set of operators or inference rules
3. Set or integrity rules

- adjacent(*G*, *x*, *y*): tests whether there is an edge from the vertices *x* to *y*;
- neighbors(*G*, *x*): lists all vertices *y* such that there is an edge from the vertices *x* to *y*;
- add_vertex(*G*, *x*): adds the vertex *x*, if it is not there;
- remove_vertex(*G*, *x*): removes the vertex *x*, if it is there;
- add_edge(*G*, *x*, *y*): adds the edge from the vertices *x* to *y*, if it is not there;
- remove_edge(*G*, *x*, *y*): removes the edge from the vertices *x* to *y*, if it is there;

Weighted Network

- get_edge_value(*G*, *x*, *y*): returns the value associated with the edge (*x*, *y*);
- set_edge_value(*G*, *x*, *y*, *v*): sets the value associated with the edge (*x*, *y*) to *v*.

Labeled Vertices

- get_vertex_value(*G*, *x*): returns the value associated with the vertex *x*;
- set_vertex_value(*G*, *x*, *v*): sets the value associated with the vertex *x* to *v*.

Information about data interconnectivity or topology is more important or as important as the data itself

Advantages:

- more natural modelling of the data which allows visualization of db-model
- queries refer to the graph structure and allow high level of abstraction without complex query rewrites
- browsing

Classical applications:

- user wants to see data connectivity
- complexity exceeds capabilities or relation db-models
- need for flexible knowledge representations
- improvements in OO-models (database and programming)

Complex networks:

- social networks
- information networks
- technological networks
- biological networks

How to delete a user? Do a traverse, delete all nodes connected with the edges, remove all edges and nodes connected with the user. This is not scalable especially with thousands and millions edges and nodes.

So operations, most of them are easy, but some of them are difficult, like the delete user operation

Data connectivity/topology:

- Natural modelling of data, allows visualization
- Queries refer to graph structure and allow high level of abstraction

Comparison to other databases:

- Physical db-models (hierarchical / Network): lack abstraction level and not flexible but navigation at record level
- Relation db-models (Codd): abstraction level, modeling has mathematical foundation
- Semantic db-models: more expressive set of semantics from users viewpoint
- Object-oriented db-models: complex data objects related to OO-programming

Possible native implementation:
Index-free adjacency

- Nodes hold pointers or references to its edges
- Traversal can be very fast
- Pointers can be fast (if…) and add no cost to traversals
- Presence of high-degree nodes make other operations very costly, because we need to look through a very long list of edges

Each node stores a list with all edges connected to it with chronological order, so we know which node to go to for every query.

therefore for the timing that this study was conducted, the conclusions are reasonable. It would be interested to see what the outcome would be if a study like that was conducted today.

Paper: Graph Analysis – Do we have to reinvent the wheel, Welc 2013

In this paper, the authors, all employed by Oracle Labs, present a thesis that SQL-based relational databases can have comparable performance to graph databases for solving graph-related problems. They also claim that the performance advantages of Neo4j do not hold in all use cases, supporting this argument with a comparative study. The study compares execution time for the calculation of the shortest path between 50 random pairs of nodes of 2 social graph, for an SQL-based relational database and 2 graph databases: Neo4j and Oracle-owned Green-Marl.

For the experiment, 2 social network datasets were used, LIVEJ and TWITTER, with a certain number of nodes and vertices. Random integer edge costs were generated for the purpose of calculating the shortest path. The most interesting fact to point out here, is that the experiment was conducted for the social network case, and not for other network applications, such as road networks, biological applications etc. As the authors pointed out, they conclude that SQL-based solution seems to perform better than Neo4j in the social network setting, however this is not the case for road networks.

The criteria that is used to evaluate the execution time of the 3 databases (SQL-based, Neo4j and Green-Marl) is Dijkstra's shortest path. There are two versions of this algorithm: uni-directional and bi-directional Dijkstra. In this experiment, for all 3 databases, a version of the bi-directional Dijkstra algorithm is used, as in all cases performs better than the uni-directional version of the algorithm. For SQL, they use the set-based version, while for Neo4j and Green-Mark a non-set-based version is used.

Conclusions
According to the presented results, the Green-Marl database seem to have the lowest execution time. However, this is due to the in-memory graph analysis. This means that the graph is already stored in the memory, therefore there is no any data-loading after the execution of the query, compared to the other 2 types of databases. The graph loading times are not included in the result graphs, therefore it cannot be compared to SQL and Neo4j approaches, where data need to be retrieved from memory after the shortest path algorithm is called. Between SQL-based relational database solution and Neo4j graph database solution, the presented results show that the former solution performs better in terms of execution time.
At this point someone has to reflect on three points. Firstly, this conlusion cannot be generalized for all network cases, as the authors themselves pointed out that SQL-based approach works better for social networks, but not for other types of networks, such as road networks. It would be preferred to run experiments in multiple types of networks in order to make a more reliable solution. Secondly, a different version of the bi-directional Dijkstra algorithm is used for the SQL-based database compared to Neo4j and Green-Mark (set-based vs non-set-based version). The authors are not clear about what is the impact on performance of that choice, therefore the results should be interpreted with some caution. Lastly, this paper is written by a team working in Oracle Labs, therefore there might be some motive for showing superior results derived by the in-house technologies vs competitive technologies.

Tip: Anything you do in RAM performs much faster than anything you perform on storage

Paper: SQL database with physical database tuning technique and NoSQL graph database comparisons, Khan 2018

4 types of NoSQL databases:
1) Key-Value: use hash tables, with unique key pointing to a particular item
2) Document databases: store data as key/value, the values of which are used to search a particular document
3) Wide-column database: hybrid architecture, use techniques and approaches from relational databases and Key-values databases are used to store the schema
4) Graph databases: store objects and relations between objects

Relational DBs:
- Vertically scalable: can process data to a certain limit
- ACID approach: atomic, consistent, isolated, durable
- Use constraints, indexes, don't store relationship information
- Use of standard query language (SQL)

No-SQL DBs:
- Horizontally scalable: can process huge amounts of data efficiently → better for big data applications
- BASE approach: basic availability, soft-state, eventually consistent
- Index-free adjacency
- Lack of standard query language
- Not many users

whenever data becomes more and more connected (large number of joins) and large in size, relational databases shown worse performance than NoSQL graph database

MongoDB performance is very well than MySQL database
RavenDB performance is 50% better than SQL Server database

→ How has all these skills/experience?

Parallel graph processing
Shared memory framework:

- SNAP
- GraphLab
- Ligra
- Green Marl

Distributed Graph Processing Frameworks:

- High level of abstraction
- Partitioned view of data and methods for read/write in and communication between partitions

Examples:

- Pregel (Google)
- Giraph (Facebook)
- GraphX (Spark)

The focus of this lecture is on the different approaches taken to implement a distributed graph processing framework

- Overview of Distributed Graph Processing, Kalavri 2018: first 3 models are the most important ones
- DGP-framework implementation of a graph algorithm (Thijs, 2017)
- GraphX as a DGP-framework

Definition (Kalavri): a distributed graph programming model is an abstraction of the underlying computing infrastructure that allows for the definition of graph data structures and the expression of graph algorithms. Data partitioning and communication is hidden.

Bulk-Synchronous Parallel
Attributes:

- A number of components, each performing processing and/or memory functions
- A router that delivers messages point to point between pairs of components
- Facilities for synchronizing all or a subset of the components at regular intervals of L time units, L: periodicity parameter. A computation consists of a sequence of supersteps.

A computation is a series of supersteps in which components:

1. Perform local computational steps
2. Transmit messages to other components
3. Receive messages as input for next superstep

Superstep is finalized after all components finish their task

Pregel – Large scale graph processing at Google
The BSP concept is ported towards networked data:
A superstar consists of tasks performed at the vertices

1. Receive messages from previous superstep
2. Execute computations and change the state of vertex and (outgoing) edges
3. Send messages to other vertices (along (outgoing) edges).

Vertices are either in Active or Inactive State

- Relation to the MapReduce paradigm
- Like the Map step, a superstar focusses first on a local action
- The reduce step is more difficult to delineate in BSP/Pregel

Paper: Pregel, A System for Large-Scale Graph Processing, Malewics et al., 2010

We built a scalable and fault-tolerant platform with an API that is sufficiently flexible to express arbitrary graph algorithms.

- Inspired by Bulk Synchronous Parallel model 9BSP)
- Consists of a sequence of iterations: supersteps
- Vertex-centric approach, like MapReduce: focus on a local action, processing each item independently, and the system composes these actions to lift computation to a large dataset.
- Distributed implementations; it doesn't expose any mechanism for detecting order of execution within a superstep, and all communication is from superstep S to superstep S + 1.
- Synchronicity: free of deadlocks and data races common in asynchronous systems.

Vertex API in C++

```
class Vertex{
      public:
         virtual void Compute(MessageIterator* msgs) = 0;
         const string& vertex_id() const;
         int64 superstep() const;
```

Paper: Kalavri 2018: first 3 models are the most important ones

- Vertex Centric
- Scatter-Gather
- Gather-Sum-Apply-Scatter
- Subgraph-centric
- Filter-process
- Graph traversals

Vertex Centric
Disadvantage of this approach: highly squid degree distribution
Many nodes will finish long before the nodes with name edges finish their processing
So the time complexity of the superstep will be equal to the time complexity of node with higher degree

Scatter Gather Approach → scatter and gather is one superstep
Each node calculates its messages, and as soon as this ready, the information spreads through the edges

Gather Sum Apply Scatter GAS
The focus is not in the node anymore but in the edge: with the distribution it focus on the edges

Challenge: computational and storage resources
Locality Sensitive Hashing: uses several random hashing functions for mapping with high probability documents with high similarity.
Documents that co-occur have high probability of being similar. The cosine similarity is approximated based on co-occurrences in buckets
The local selection functions happen at the level of individual nodes

Similarity depends on the total number of references and the references that co-occur

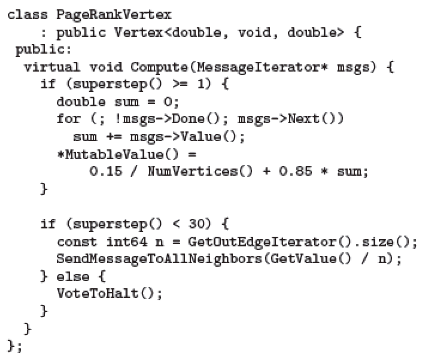
Netflix dataset:

- Many people watch the same movie, but those people are not similar
- For movies with less viewers, then the probability that those viewers are similar is higher

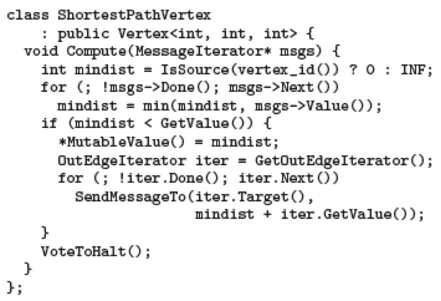
Vertex-cut: we cut nodes and distribute
We have to copy the info of each part to a particular partition.

How to improve the sum step: if you know that particular nodes and close to each other, we could distribute the edges having a kind of clustering/community algorithm, so those nodes are together in one partition. We need a continuous vertex-cut while doing our job.

Pregel: 2 common graph algorithms



```
class PageRankVertex
    : public Vertex<double, void, double> {
 public:
  virtual void Compute(MessageIterator* msgs) {
    if (superstep() >= 1) {
      double sum = 0;
      for (; !msgs->Done(); msgs->Next())
        sum += msgs->Value();
      *MutableValue() =
          0.15 / NumVertices() + 0.85 * sum;
    }

    if (superstep() < 30) {
      const int64 n = GetOutEdgeIterator().size();
      SendMessageToAllNeighbors(GetValue() / n);
    } else {
      VoteToHalt();
    }
  }
};
```



```
class ShortestPathVertex
    : public Vertex<int, int, int> {
  void Compute(MessageIterator* msgs) {
    int mindist = IsSource(vertex_id()) ? 0 : INF;
    for (; !msgs->Done(); msgs->Next())
      mindist = min(mindist, msgs->Value());
    if (mindist < GetValue()) {
      *MutableValue() = mindist;
      OutEdgeIterator iter = GetOutEdgeIterator();
      for (; !iter.Done(); iter.Next())
        SendMessageTo(iter.Target(),
                      mindist + iter.GetValue());
    }
    VoteToHalt();
  }
};
```

Review GraphX (Spark)

- Does their Pregel implementation follow the genuine Pregel model?
- How many functions are required for the implementation of the Pregel function
- How do they achieve a fair distribution of task executions?

Resilient distributed graphs in GraphX in Spark

- Computations beyond Map-reduce
- In-memory storage abstraction (Resilient Distributed Datasets)
- Integrated into Scala, Python and R
- Optional data partitioner

Core data structure = Immutable Graph

- Directed adjacency structure
- User defined attributes associated with vertices and edges
- Each transformation results in a new graph

Partitioning in GraphX

1. This paper uses a messaging procedure and some calculation functions. Which programming model is applied?

It uses Pregel messaging framework, built on the Bulk Sunchronous Parallel model, and it's implementing a sequence of supersteps. This is a vertex centric model.

2. What can you say about memory consumption of this algorithm?

It's very large because it is an all-nearest neighbor problem in a huge feature space. Therefore everything should be stored in memory at some point.

- Vertex centric
- All nearest-neighbors

Lecture 6: Random Networks

This lecture will study random models that have been introduced
We can study theoretical properties of real world graphs
Compare properties with real observations
Create realistic test graphs which enables development

Three random graph model

1. Erdos-Renyi Model (1959)
   - Fixed number of vertices and edges
   - Any topology of graph is equally likely

   → Creating a Erdos-Reyni network with igraph-python:
   - er = graph.Erdos_Renyi(200, 0.02)
     - 200: number of nodes
     - 0.02: probability of any nodes to be connected
   - er = graph.Erdos_Renyi(200, m=3000)
     - m=3000 : 3000 edges to be distributed between the 200 nodes

   Gilbert: For any given pair of nodes, the existence of an edge between them has the same probability

2. Watts-Strogatz Model (1998)
   - Start from a ring, all nodes are connected to their networks
   - Fixed number of vertices and edges in a regular ring lattice
   - Rewiring of edges with given probability, the target node of an edge is changed to another node

   → Creating a Watts-Strogatz network with igraph-python:
   - ws = graph.Watts_Strogatz(1, 200, 2, 0.1)
     - 200: number of nodes
     - 2 : for each node we connect to a neighborhood of 2
     - 0.1 : rewiring probability of 10%

3. Barabasi-Albert Model (1999): focuses on the growth of the network
   - The number of nodes increases over time
   - Each new node comes with additional (n) edges
   - The degree of existing nodes determines the probability to become target of additional edge
   - The source node it given, but target node is not given, and has different probability depending on the degree of the nodes

   → Creating a Barabasi-Albert network with igraph-python:
   - ba = graph.Barabasi(200, m=2) or m=16
     - 200 nodes
     - m=2 : each new node brings 2 edges with it
     - for m=2, number of edges is 397, because the first node comes with zero edges and the second node can only connect with the first node

Question 1: which random model is used for this graph?

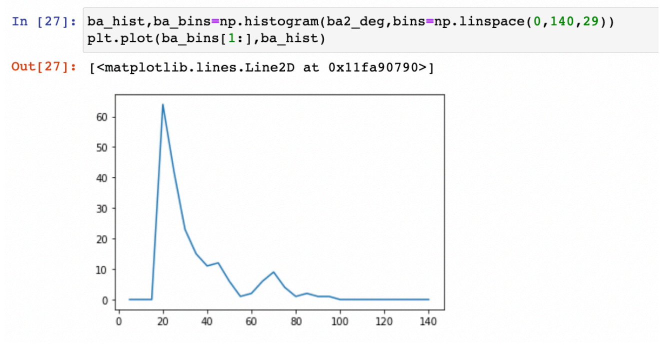
$$P(\deg(v) = k) = \binom{n-1}{k} p^k (1-p)^{(n-1-k)}$$

- For large n, distribution becomes a Poison distribution

$$P(\deg(v) = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

2. Watts-Strogatz Model
   - Distribution becomes Poisson when rewiring probability close to 1

Question 4: What is the degree distribution of a BA-model?



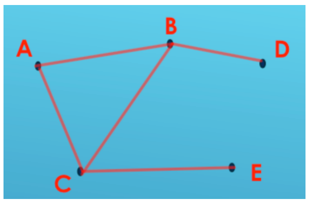
```
In [27]: ba_hist,ba_bins=np.histogram(ba2_deg,bins=np.linspace(0,140,29))
         plt.plot(ba_bins[1:],ba_hist)

Out[27]: [<matplotlib.lines.Line2D at 0x11fa90790>]
```

Scale free, power law

$$P(\deg(v) = k) = k^{-\lambda} \text{ with } \lambda \approx 3$$

Global clustering (Transitivity)

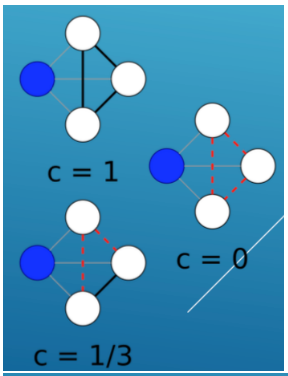
$$C = \frac{3 * number\ of\ triangles}{number\ of\ connected\ triplets}$$



Triangles = ABC

Connected triplet = A-B-C, B-A-C, A-C-B, C-B-D, A-B-D, A-C-E, B-C-E

Local clustering



$$C = \frac{2\left|\{e_{jk}\colon v_j, v_k \in N_i, e_{jk} \in E\}\right|}{d_i(d_i - 1)}$$

Question 5: Which model shows the highest clustering? Watts-Strogatz

The shortest path of the network is estimated by sampling a fraction of all nodes, determining their distances from all other points in the network, and averaging over all pairs of nodes. In both M and NS networks, the shortest path length is decreasing with time, as mode nodes and links are added to the network. This might be caused by 2 possible reasons:

- New papers that are added, might connect already existing authors with more authors, therefore the interconnectivity is increasing
- The dataset is not complete (only paper of 8 years). In a complete dataset we could expect separation to increase over time, because new authors appear with low interconnectivity with the rest of the community

3. Discuss the properties of the global and local clustering in both networks.
4. Can you explain the relation between the evolution of the degree distribution, separation and the clustering coefficient? Especially the latter two show a particular pattern over time. Is this what you would expect?
5. Can you explain what the roll of preferential attachment is for both new and established authors in the collaboration networks?

Lecture 7: Cluster Analysis

Clustering is a general term used in analytics. It refers to a broad set of techniques that share a common goal: to create sets or groups (so called clusters) of items or objects which are similar or share similar properties but where the structure of the data is hidden or unknown. These techniques are also described as approaches within unsupervised learning.
Fundamental hypothesis or assumption: the cluster structure is uniquely encoded in the observed data set: ground truth

Each cluster approach has to solve common issues:

1. how to define similarity or sharing of similar properties between objects and between groups
2. which criterion is used to assign two objects to the same cluster or to divide them from each other
3. how many clusters or groups are required to represent the structure in the data
4. can items or objects belong to only one group or is multiple assignment possible
5. when is a clustering of items a good clustering

Important distinction: is the number of clusters known??

- Hierarchical
    1. Agglomerative vs divisive
    2. Production of dendrogram
    3. Distance or similarity measure
    4. Linkage rules
- K-means
    1. Have k centroids or means (random or based on a priori knowledge)
    2. Distance or similarity measures
    3. Linkage rules
- Linkage criteria
    1. Maximum or complete-linkage clustering
    2. Minimum or single-linkage clustering
    3. Mean or average linkage clustering
    4. Centroid linkage clustering
    5. Minimum energy clustering
- Most common methods in Unsupervised Learning
    1. Spectral clustering (using eigenvalues)
    2. (Gaussian) Mixture models
    3. Self-organizing maps
    4. Hidden Markov models

How to adapt for graph data?
Use matrix representation of network

- Adjacency matrix / incidence matrix
- Laplacian matrix L=D-A

Lij = { 0 if no edge between i and j
{-1 if I =/ j and the edge ij exist
{ degree of vi if i=j

Newman & Girvan introduce a new measure for the goodness (quality) of a clustering: more edges inside a cluster that outside (locally dense connected subgraph)
Modularity = fraction of edges within the clusters minus the expected fraction if edges were distributed randomly

$$Q=\sum_i (e_{ii}-a_i^2)=\mathrm{Tr}\ \mathbf{e}-\|\mathbf{e}^2\|,$$

e: share of edges over the total graph / adjacency matrix
a: sum over the rows
then we find a^2 and e-a^2
Modularity: sum of e-a^2 of all clusters

Flow betweenness: inspired by the elementary circuit theory, more than a single unit of flow can be carried between source and target by making simultaneous use of several different paths through the network, those with least resistance carrying the greatest fraction of the flow. The flow betweenness of an edge is the absolute value of the flow along the edge summed over all source/target pairs. Time complexity: O(n+m)mn^2) on regular graphs, or O(n^4) on sparse graphs.

Random-walk betweenness: the expected number of times that a random walk between a particular pair of vertices will pass down a particular edge and sum over all vertex pairs. Time complexity: same as flow betweenness

In practice, the author note that for the applications examined in the paper, the choice of type of betweenness is unimportant, as they actually give rather similar results. They also recommend for most problems to use the shortest path betweenness.

3. What is your opinion about time and space complexity of the proposed approach

Space complexity

<u>Paper: Fast unfolding of communities in large networks, Blondel et al, 2008</u>
<u>→ Louvain Algorithm</u>

1. The authors take a hierarchical approach with an unknown number of clusters at the start of the procedure. Is their approach agglomerative or divisive?

The authors follow the agglomerative approach. The algorithm starts with assigning a different community to each node of the network. Then for each node i, its neighbors are considered and then the gain of modularity is calculated for i getting removed from each own cluster and be placed in the community of all different j_s. Finally the node is getting placed in the community of j for which the gain is maximized. This process is repeated for all clusters several times until modularity reaches its maxima. Then we obtain a new network, where nodes are the detected communities and weights of the links correspond to the sum of the weights of the links between the nodes of the corresponding communities. Then the first part of the algorithm runs again and detect new communities of the initial communities. The algorithm runs for several iterations and converges when the global maximum of modularity is found, resulting in a hierarchical structure of communities and subcommunities in the network.

2. How are *cuts/merges* introduced?
3. How are *links between clusters* calculated?
4. What is your opinion about the time and space complexity

Regarding time complexity, it is linear to the size of the network. Judging from the time needed to detect communities in large networks, such as the WebBase 2001 network, the algorithm seems to be rather fast. The efficiency of the algorithm derives from the formula used to calculate the modularity gain when a node i is merged or removed by a community. Moreover, the number of passes tends to be small, smaller than 5 in all cases. The first pass always is the longest as the initial number of communities equals to the number of nodes. The following passes are significantly faster. There are some improvements in computational time when ordering of the nodes is used, however it remains reasonably small.

The storage of the network in the main memory is where the limitation of this algorithm lies. The memory is cubical for number of nodes.

5. What is your opinion about *resolution limit*? Does this approach suffer from this problem?

Resolution limit refers to the limitation of community detection algorithms to identify smaller communities in a network, as they usually get absorbed by larger communities.
This algorithm seems to suffer less from this problem due to its hierarchical/agglomerative approach. It is possible that in the last pass of the algorithm, smaller communities have been merged with larger ones, however we can always review the results of intermediate passes by zooming in the network and observe its structure with the desired resolution. We can also verify the results of the intermediate passes, by considering the final communities as an independent network and applying the same algorithm again.

<u>Paper: Performance of modularity maximization in practical contexts, Good et al., 2009</u>
reduce or coarse-grain a system by dividing global heterogeneity into relatively homogeneous substructures
a “good” partition—with *Q* closer to unity—identifies groups with many more internal connections than expected at random
a “bad” partition—with *Q* closer to zero—identifies groups with no more internal connections than we expect at random
optimal partition: maximizes modularity
Resolution limit: could be seen as a consequence of assuming that intermodule connections follow a random graph model, which induces an explicit preference on the weight of between-module connections. But better explained as a systematic deviation between the intermodule connectivity *eij* and the connectivity expected under the random- graph null model �*ei j*� = *didi* / 2*m*,

1. What are the basic assumptions often underlying modularity maximization approaches discussed in this paper

i) empirical networks with modular structure tend to exhibit a clear optimal partition
ii) high-modularity partitions of an empirical network are structurally similar to this optimal partition
iii) the estimated *Q*max can be meaningfully compared across networks

Modularity based approaches - Improvements:

- Add resolution parameter $$\mathcal{H} = \frac{1}{2m}\sum_c \left(e_c - \gamma\frac{K_c^2}{2m}\right),$$
    1. Gamma: scale the expected number of links, it influences the resolution, can get a more detailed view of the clustering.
    2. Problem: applied to the total network, and not a subpart of it, so it will influence the complete network
- Other Quality Function Constant Potts Model (CPM)

Information based approach:
Alternative idea for good clustering: The description length of a random walk across nodes and modules is lower when using an optimal partitioning of the network

- quality measure based on descriptive length of a random walk across the network.
- A random walk is going through the network, noting down the nodes and edges that it is passing by.
- The nodes are given codes, and create a kind of codebook
    1. The nodes that are passing by often get a shorter codes
    2. Nodes that are next to the other get a same part of the code, indicating that they are in the same neighborhood
    3. So there is codebook for the clusters and codebook for the nodes

Map Equation
www.mapequation.org

$$L(M) = q_{\curvearrowright}H(Q) + \sum_{i=1}^{m} p_{i\circlearrowright}H(P_i)$$

- This function expresses the theoretical minimum length of the description of random walk in a network with partitioning M
- The objective is to minimize this function
- The first term refers to transfers from one module to another, code used to indicate that a code leaves a cluster and goes to another one
- The second term refers to description of the walk inside m different modules

Theoretical minimum length

$$L(P) = H(P) \equiv -\sum_i p_i \log p_i$$

- Shannon's source coding theorem is used to express this minimum length as the entropy of the probability distribution
- Probability distributions can be calculated for the transition from one module to another and for nodes within a module
- The partitioning into modules allows the use to different codebooks (Index & Module)

Probability distributions using random walks

- Probability of step from a to b $$p_{\alpha\to\beta} = \frac{w_{\alpha\to\beta}}{\sum_\beta w_{\alpha\to\beta}}.$$
- Probability of being at node a $$p_\alpha = \sum_\beta p_\beta p_{\beta\to\alpha}.$$
- Adding a teleportation parameter proportional to the total weights of links from the node

$$p^*_\alpha = (1-\tau)\sum_\beta p^*_\beta p_{\beta\to\alpha} + \tau\frac{\sum_\beta w_{\alpha\to\beta}}{\sum_{\alpha,\beta} w_{\beta\to\alpha}}.$$

- Probabilities for links and nodes $$q_{\beta\to\alpha} = p^*_\beta p_{\beta\to\alpha}$$ $$p_\alpha = \sum_\beta q_{\beta\to\alpha}.$$

Probability distribution of modules/clusters

- Probability entering or exiting a module $$q_{i\curvearrowleft} = \sum_{\alpha\in j\neq i,\beta\in i} q_{\alpha\to\beta}$$ $$q_{i\curvearrowright} = \sum_{\alpha\in i,\beta\in j\neq i} q_{\alpha\to\beta}.$$

- Probability of using Index Codebook $q_{\curvearrowright} = \sum_{i=1}^{m} q_{i\curvearrowright}$
- Probability of using Module Codebook of module i $p_{i\circlearrowright} = \sum_{\alpha\in i} p_\alpha + q_{i\curvearrowright}$

Calculating Entropy

e) Individual (aggregated) nodes are moved to optimize partition

So as soon as the clusters as refined, the algorithm runs again at each individual clusters in order to finale the communities within the larger clusters. We get a layered clustering solution.

The refinement phase in the Leiden algorithm

a) Only within each community of initial partition
b) Each node in the community start as its own partition
c) Merges are performed
    1. Increase of quality function
    2. Sufficiently connected
    3. Not greedy (not highest increase but probability relative to increase) → it solved the degeneracy problem

Optimization of local moving in the Leiden algorithm

a) Only nodes whose neighborhood has changed is visited
b) Each node is in queue (random order)
c) Node is taken, processed and if changes, neighborhood is added to the queue

Vs Louvain: all nodes are revisited, here only the nodes whose neighborhood has changed is visited.

Paper: From Louvain to Leiden: guaranteeing well-connected communities

3 phases:

1) Local moving phase: individual nodes are moved to the community that yields the largest increase in the quality function
2) Refinement of the partition: it merges local nodes within a community to subcommunities, resulting to P_{refine}
3) Aggregation of network: an aggregate network is created based on the partition obtained in the refinement phase. Each community in this partition becomes a node in the aggregate network. The two phases are repeated until the quality function cannot be increased further.

    - smart local move
    - fast local move
    - random neighbour move

Guarantees:
Γ: resolution parameter

1. All communities are γ-separated.
2. All communities are γ-connected.
3. All nodes are locally optimally assigned.
4. All communities are subpartition γ-dense.
5. All communities are uniformly γ-dense.
6. All communities are subset optimal.

1. What is your opinion about the resolution limit for the Leiden algorithm?

The Leiden algorithm proposed a new linkage procedure that addresses the degeneracy problem, where different assignments of nodes to clusters result to same modularity score, causing a plateau. The resolution limit challenge can be addressed by choosing an appropriate function to optimize. For example, using Leiden algorithm with map function will be more effective in tackling the resolution limit compared to Leiden algorithm with modularity optimization.

Moreover, Leiden algorithm has the property of when finding subcommunities in a community, it splits it up, guaranteeing an increase in modularity.

2. Figure 4. Indicates that the iterations are important for the improvements of the Leiden algorithm. The share of badly connected communities decreases with each iteration. Provide an explanation why this algorithm has a higher share of badly connected communities than the Louvain method in the first iteration.

The Louvain algorithm is initialized with a cluster for each individual node, then nodes are merging with their neighboring nodes and join their cluster, given an increase in modularity. The first iteration of the Louvain algorithm is the longest one as all individual nodes are forming clusters with each other, usually from the same neighborhood and connected to each other. This is why the number of disconnected clusters during the first iteration of Louvain method is rather low.

On the other hand, Leiden algorithm in the first iteration applies the local moving phase which results to a partition P. According to Waltman L. et al (2013), each node is assigned to its own singleton community and then a local moving heuristic is applied. In some cases, this yields a community structure of one big cluster that includes all nodes, in some other cases, it results to multiple communities that include some of the nodes in the subnetwork. Then the aggregated network is created based on partition P and then the refinement phase follows. The reason why there is a higher share of badly connected communities could lie in the output of the local moving phase, where initial detected communities need to be further split up in the following iterations.