UNIT II

UNIT 2 SOCIAL NETWORK ANALYSIS SOFTWARE, TOOLS AND LIBRARIES

Modelling and aggregating social network data: Ontological representation of social individuals – Ontological representation of social relationships - Aggregating and reasoning with social network data – Advanced representations. Social network analysis software - Tools - Libraries .

2.1 MODELLING AND AGGREGATING SOCIAL NETWORK DATA

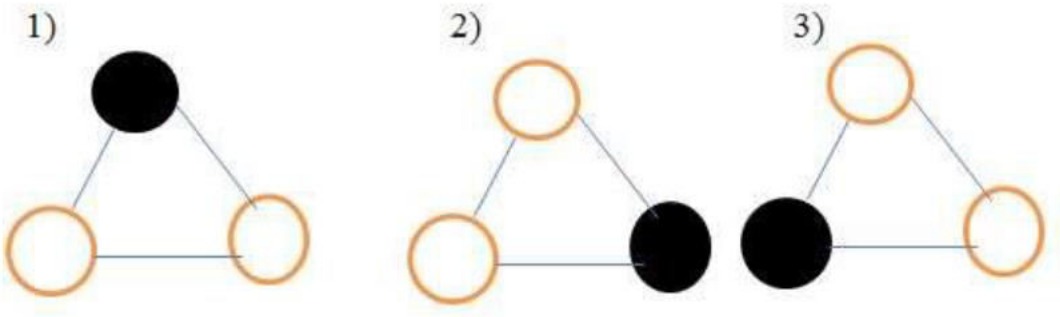
- The most common kind of social network data can be modeled by a graph where the nodes represent individuals and the edges represent binary social relationships. (Less commonly, higher-arity relationships may be represented using hyper-edges, i.e. edges connecting multiple nodes.)
- Additionally, social network studies build on attributes of nodes and edges, which can be formalized as functions operating on nodes or edges.
- A number of different, proprietary formats exist for serializing such graphs and attribute data in machine-processable electronic documents.
- The most commonly encountered formats are those used by the popular network analysis packages **Pajek and UCINET**. These are text-based formats which have been designed in

  a way so that they can be easily edited using simple text editors.
- Unfortunately, the two formats are incompatible. Further, researchers in the social sciences often represent their data initially using Microsoft Excel spreadsheets, which can be exported in the simple CSV (Comma Separated Values) format.
- The GraphML format represents an advancement over the previously mentioned formats in terms of both interoperability and extensibility.
- GraphML originates from the information visualization community where a shared format greatly increases the usability of new visualization methods.
- GraphML is therefore based on XML with a schema defined in XML Schema. This has the advantage that GraphML files can be edited, stored, queried, transformed etc. using generic XML tools.
- Common to all these generic graph representations is that they focus on the graph structure, which is the primary input to network analysis and visualization.
- Attribute data when entered electronic form is typically stored separately from network data in Excel sheets, databases or SPSS tables.

2.2 RANDOM WALKS AND THEIR APPLICATIONS

A Random Walk in synthesis:

- Given an indirected graph and a starting point, select a neighbour at random
- Move to the selected neighbour and repeat the same process till a termination condition is verified
- The random sequence of points selected in this way is a random walk of the graph



Important parameters of random walk:

- **Access time or hitting time:** H_{ij} is the expected number of steps before node j is visited, starting from node i
- **Commute time**: i → j → i: $H_{ij} + H_{ji}$
- **Cover time**: Starting from a node/distribution the expected number of steps to reach every node.

Applications of Random Walks on Graphs

- Ranking Web Pages
- HITS on citation network
- Clustering using random walk

2.3 USE OF HADOOP AND MAP REDUCE

Map reduce

- Data-parallel programming model for clusters of commodity machines
- Pioneered by Google
  - Processes 20 PB of data per day
- Popularized by open-source Hadoop project
  - Used by Yahoo!, Facebook, Amazon, …

Map Reduce used for

- At Google:

1. Index building for Google Search
2. Article clustering for Google News
3. Statistical machine translation

- At Yahoo!:

1. Index building for Yahoo! Search
2. Spam detection for Yahoo! Mail
- At Facebook:

1. Data mining
2. Ad optimization
3. Spam detection

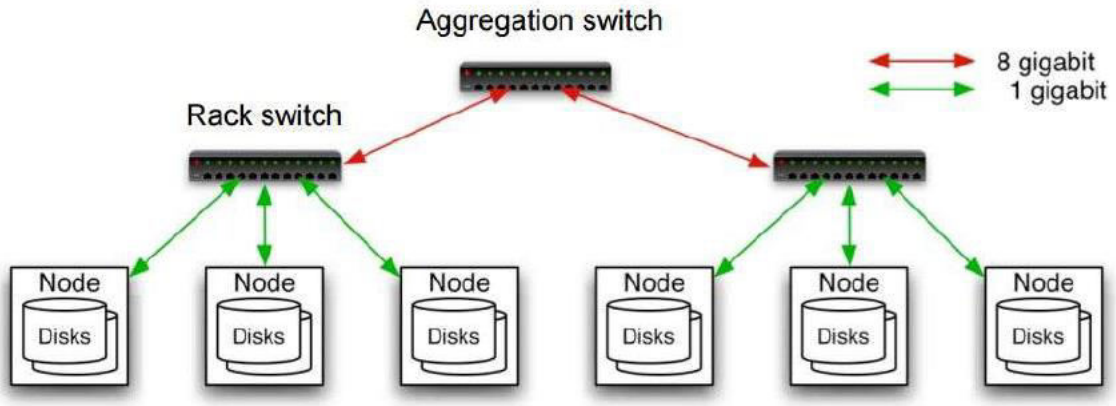
In research:
- Analyzing Wikipedia conflicts (PARC)
- Natural language processing (CMU)
- Bioinformatics (Maryland)
- Particle physics (Nebraska)
- Ocean climate simulation (Washington)

Map Reduce Goals

1. Scalability to large data volumes:
- Scan 100 TB on 1 node @ 50 MB/s = 24 days
- Scan on 1000-node cluster = 35 minutes

2. Cost-efficiency:
- Commodity nodes (cheap, but unreliable)
- Commodity network
- Automatic fault-tolerance (fewer admins)
- Easy to use (fewer programmers)

TYPICAL HADOOP CLUSTER:



40 nodes/rack, 1000-4000 nodes in cluster
- 1 GBps bandwidth in rack, 8 GBps out of rack

- Node specs (Yahoo! terasort): 8 x 2.0 GHz cores, 8 GB RAM, 4 disks (= 4 TB?)

Challenges

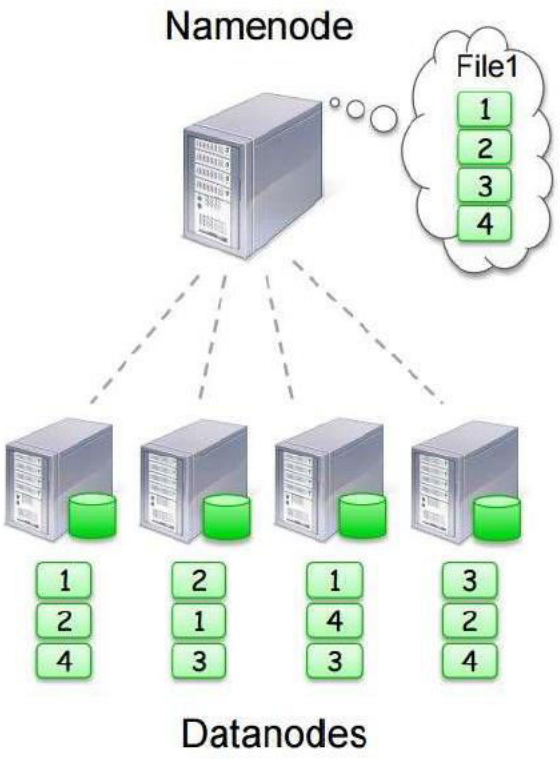
- Cheap nodes fail, especially if you have many
  - Mean time between failures for 1 node = 3 years
  - MTBF for 1000 nodes = 1 day
  - Solution: Build fault-tolerance into system
- Commodity network = low bandwidth
  - Solution: Push computation to the data
- Programming distributed systems is hard
  - Solution: Users write data-parallel "map" and "reduce" functions, system handles work distribution and faults

Hadoop Components:

- Distributed file system (HDFS)
  - Single namespace for entire cluster
  - Replicates data 3x for fault-tolerance
- MapReduce framework
  - Executes user jobs specified as "map" and "reduce" functions
  - Manages work distribution & fault-tolerance

Hadoop Distributed File System:

- Files split into 128MB blocks
- Blocks replicated across several data nodes (usually 3)
- Namenode stores metadata (file names, locations, etc)
- Optimized for large files, sequential reads
- Files are append-only



MapReduce Programming Model:

- Data type: key-value records
- Map function:

$$(K_{in}, V_{in}) \rightarrow \text{list}(K_{inter}, V_{inter})$$

- Reduce function:

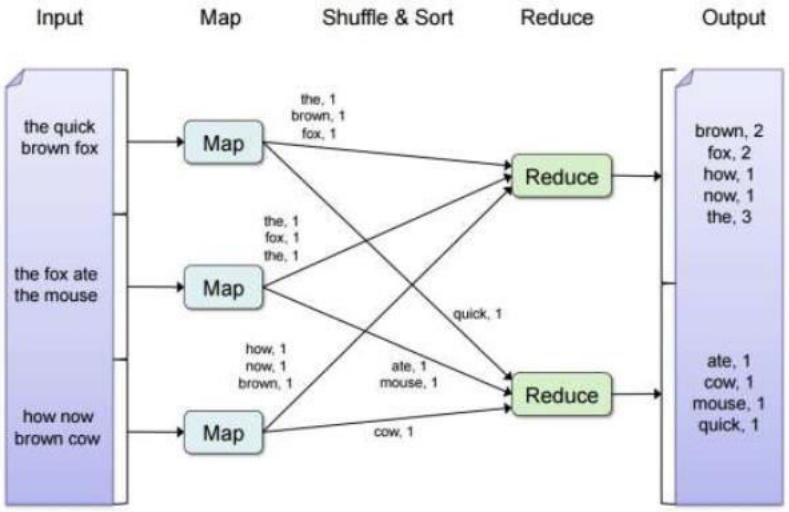
$$(K_{inter}, \text{list}(V_{inter})) \rightarrow \text{list}(K_{out}, V_{out})$$

Example: Word Count:

```
def mapper(line):
  foreach word in line.split():
    output(word, 1)

def reducer(key, values):
  output(key, sum(values))
```

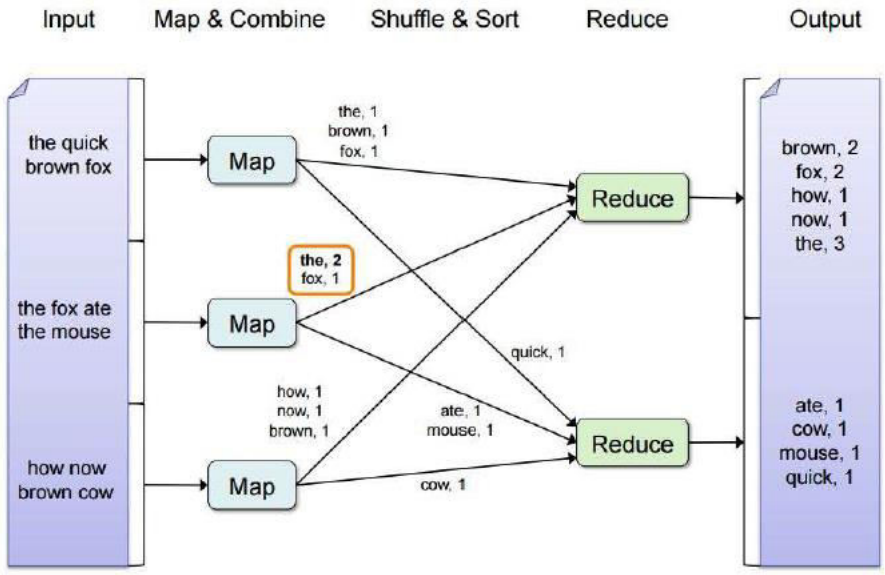
Word Count Execution:



An Optimization: The Combiner

- Local aggregation function for repeated keys produced by same map
- For associative ops. like sum, count, max
- Decreases size of intermediate data

Word Count with Combiner



MapReduce Execution Details:

- Mappers preferentially placed on same node or same rack as their input block
  - Push computation to data, minimize network use
- Mappers save outputs to local disk before serving to reducers
  - Allows having more reducers than nodes
  - Allows recovery if a reducer crashes

Fault Tolerance in MapReduce:

- If a task crashes:
  - Retry on another node
- OK for a map because it had no dependencies
- OK for reduce because map outputs are on disk
  - If the same task repeatedly fails, fail the job or ignore that input block
- If a node crashes:
  - Relaunch its current tasks on other nodes
  - Relaunch any maps the node previously ran
- Necessary because their output files were lost along with the crashed node
- If a task is going slowly (straggler):
  - Launch second copy of task on another node
  - Take the output of whichever copy finishes first, and kill the other one

2.4 ONTOLOGICAL REPRESENTATION OF SOCIAL INDIVIDUALS AND RELATIONSHIPS

2.4.1 ONTOLOGICAL REPRESENTATION OF SOCIAL INDIVIDUALS

(i) The Friend-of-a-Friend (FOAF) ontology that we use in our work is an OWL based format for representing personal information

(ii) FOAF started as experimentation with Semantic Web technology.

(iii) The idea of FOAF was to provide a machine processable format for representing the kind of information that made the original Web successful, namely the kind of personal information described in homepages of individuals.

(iv) Thus FOAF has a vocabulary for describing personal attribute information typically found on homepages such as name and email address of the individual, projects, interests, links to work and school homepage etc.

(v) FOAF profiles contain a description of friends the using the individuals same vocabulary that is used to describe the individual himself.

(vi) FOAF became the center point of interest in 2003 with the spread of Social Networking Services such Friendster, Orkut, LinkedIn etc.

Drawbacks:

1. The information is under the control of the database owner
2. Centralized systems do not allow users to control the information they provide on their own terms.

(vii) FOAF profiles are created and controlled by the individual user and shared

in a distributed fashion. FOAF profiles are typically posted on the personal website of the user and linked from the home page user switch the HTML META tag.

(viii) An advantage of FOAF in terms of sharing FOAF data is the relative stability of the ontology. The number of FOAF users means that the maintainers of the ontology are obliged to keep the vocabulary and its semantics stable.

FOAF Basics
Agent
Person
name
nick
title
homepage
mbox
mbox_sha1sum
img
depiction (depicts)
surname
family_name
givenname
firstName

Personal Information
weblog
knows
interest
currentProject
pastProject
plan
based_near
workplaceHomepage
workInfoHomepage
schoolHomepage
topic_interest
publications
geekcode
myersBriggs
dnaChecksum

Online Accounts / IM
OnlineAccount
OnlineChatAccount
OnlineEcommerceAccount
OnlineGamingAccount
holdsAccount
accountServiceHomepage
accountName
icqChatID
msnChatID
aimChatID
jabberID
yahooChatID

Projects and Groups
Project
Organization
Group
member
membershipClass
fundedBy
theme

Documents and Images
Document
Image
PersonalProfileDocument
topic (page)
primaryTopic
tipjar
sha1
made (maker)
thumbnail
logo

For example, the SIOC (Semantically Enabled Online Communities) project aims at connecting discussions across various types of for a Usenet, discussion boards, blogs, mailing lists etc by exposing the postings according to a shared ontology.

The key concepts of this ontology are the sioc:User account that is used to create a sioc:Post, which is part of a sioc:Forum at a certain sioc:Site. A sioc: User is not a subclass of foaf :Person (as a person may have multiple accounts), but related to the description of a person using the sioc:account of property. While FOAF has a rich ontology for characterizing individuals—especially with respect to their online presence—, but it is rather poor as a vocabulary for describing relationships.

2.4.2 ONTOLOGICAL REPRESENTATION OF SOCIAL RELATIONSHIPS

Ontological representations of social networks such as FOAF need to be extended with a framework for modeling and characterizing social relationships for two principle reasons:

(1) To support the automated integration of social information on a semantical basis and
(2) To capture established concepts in Social Network Analysis.

Characteristics of social relationships

- **Sign**: A relationship can represent both positive and negative attitudes such as like or hate. The positive or negative charge of relationships is the subject of balance theory
- **Strength:** Tie strength itself is a complex construct of several characteristics of social relations. Tie strength lists the following: Frequency/frequent contact , Reciprocity, Trust/enforceable trust, Complementarity, Accommodation/adaptation, Indebtedness/imbalance, Collaboration, Transaction investments, Strong history, Fungible skills, Expectations, Social capital
- **Provenance**: A social relationship may be viewed differently by the individual participants of the relationship, sometimes even to the degree that the tie is unreciprocated. Similarly, outsiders may provide different accounts of the relationship, which is a well-known bias in SNA.
- **Relationship history**: Social relationships come into existence by some event involving two individuals
- **Relationship roles**: A social relationship may have a number of social roles associated with it, which we call relationship roles. For example, in a student/professor relationship within a university setting there is one individual playing the role of professor, while another individual is playing the role of a

student. Both the relationship and the roles may be limited in their interpretation and use to a certain social context.

Ideally, all users of all these services would agree to a single shared typology of social relations and shared characterizations of relations. However, this is neither feasible nor necessary. What is required from such a representation is that it is minimal in order to facilitate adoption and that it should preserve key identifying characteristics such as the case of identifying properties for social individuals.

Conceptual model

→Social relations could be represented as n-ary predicates; however, n-ary relations are not supported directly by the RDF/OWL languages. There are several alternatives to n-ary relations in RDF/ OWL

→In all cases dealing with n-ary relations we employ the technique that is known as *reification*: we represent the relation as a class, whose instances are concrete relations of that type.

→ One may recall that RDF itself has a reified representation of statements: the *rdf :Statement* object represents the class of statements.

→This class has three properties that correspond to the components of a statement, namely ***rdf: subject*, *rdf* :*predicate*, *rdf* :*object*.**

→These properties are used to link the statement instance to the resources involved in the statement.

→In other words relationships become subclasses of the ***rdf* :*Statement*** class. Common is that the new Relationship class is related to a general Parameter class by the **hasParameter** relationship. Relationship types such as Friendship are subclasses of the Relationship class, while their parameters (such as strength or frequency) are subtypes of the Parameter class.

Two alternatives:

→The **first scheme** borrows from the design of OWL-S for representing service parameters, as used in the specification of the profile of a Web Service. Here, parameters are related by the valued-by metaproperty to their range. For example in an application Strength may be a subclass of Parameter valued-by integers. The **disadvantage** of this solution is that specifying values requires two statements or the introduction of a

constructed property.

→ The **second alternative** differs in that the "native "representing meth parameters: the generic Parameter class is defined as a subclass of *rdf :Property*. This model has the advantage that it becomes more natural to represent parameter values and restrictions on them. The **disadvantage** is that this solution is not compliant with OWL.

DL Social relations are socially constructed objects: they are constructed in social environments by assigning a label to a common pattern of interaction between individuals.

Cognitive structuring, works by applying the generic pattern we associate with such a relationship to the actual state-of-affairs we observe. For example, a student/professor relationship at the Free University of Amsterdam is defined by the social context of the university and this kind of relationship may not be recognizable outside of the university.

The below figure shows descriptions and Situations ontology design pattern that provides a model of context and allows to clearly delineate these two layers of representation.

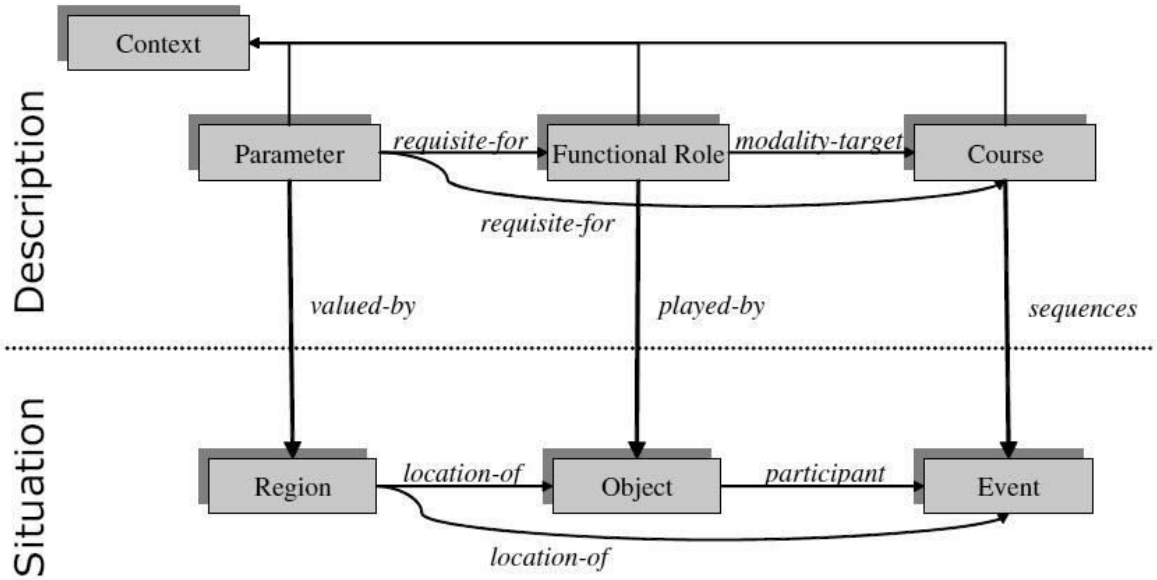


Fig. The Descriptions and Situations ontology design pattern

D&S is a generic pattern for modeling non-physical objects whose intended meaning results from statements, i.e. it emerges in combination with other entities. For example, a norm, a plan, or a social role is usually represented as a set of statements and not as a concept.

D&S is an ontology-design pattern in the sense that it is used as a template for creating domain ontologies in complex areas. D & S has been successfully applied in a wide range of real-life ontology engineering projects from representing Service Level Agreements (SLAs) to the descriptions of Web Services.

2.5 AGGREGATING AND REASONING WITH SOCIAL NETWORK DATA

2.5.1 ADVANCED REPRESENTATIONS

EXTRACTING EVOLUTION OF WEB COMMUNITY FROM A SERIES OF WEB ARCHIVE

The extraction of Web community utilizes Web community chart A graph of communities, in which related communities are connected by weighted edges. The main advantage of the Web community chart is existence of relevance between communities.

2.5.1 **Notations Used**

- $t1$, $t2$, ..., tn: Time when each archive crawled. Currently, a month is used as the unit time
- $W(tk)$: The Web archive at time tk
- $C(tk)$: The Web community chart at time tk
- $c(tk)$, $d(tk)$, $e(tk)$, ...: Communities in $C(tk)$

Types of Changes

- ✓ **Emerge** A community $c(tk)$ emerges in $C(tk)$, when $c(tk)$ shares no URLs with any community in $C(tk-1)$.

- ✓ **Dissolve**

A community $c(tk-1)$ in $C(tk)$ has dissolved, when $c(tk-1)$ shares no URLs with any community in $C(tk)$

- ✓ **Growth and Shrink**

The community grows when new URLs are appeared in $c(tk)$, and shrinks when URLs disappeared from $c(tk-1)$.

- ✓ **Split**

c(tk−1) shares URLs with multiple communities in $C(tk)$

- ✓ **Merge**

When multiple communities ($c(tk-1)$), $d(tk-1)$, ...) share URLs with a single community $e(tk)$, these communities are merged into $e(tk)$

Evolution Metrics

Evolution metrics measure how a particular community c(tk) has evolved. The metrics are defined by differences between c(tk) and its corresponding community c(tk−1).

Growth Rate

The growth rate, Rgrow(c(tk−1), c(tk)), represents the increase of URLs per unit time. It allows us to find most growing or shrinking communities.

$$R_{grow}(c(t_{k-1}), c(t_k)) = \frac{N(c(t_k)) - N(c(t_{k-1}))}{t_k - t_{k-1}}$$

Stability

Represents the amount of disappeared, appeared, merged and split URLs per unit time. A stable community on a topic is the best starting point for finding interesting changes around the topic.

$$R_{stability}(c(t_{k-1}),c(t_k)) = \frac{N(c(t_k)) + N(c(t_{k-1})) - 2N_{sh}(c(t_{k-1}),c(t_k))}{t_k - t_{k-1}}$$

Disappearance rate

The number of disappeared URLs from c(tk−1) per unit time. Higher disappear rate means that the community has lost URLs mainly by disappearance.

$$R_{disappear}(c(t_{k-1}),c(t_k)) = \frac{N_{dis}(c(t_{k-1}))}{t_k - t_{k-1}}$$

Merge rate

The number of absorbed URLs from other communities by merging per unit time. Higher merge rate means that the community has obtained URLs mainly by merging.

$$R_{merge}(c(t_{k-1}),c(t_k)) = \frac{N_{mg}(c(t_{k-1}))}{t_k - t_{k-1}}$$

Split Rate

The split rate, Rsplit (c(tk−1), c(tk)), is the number of split URLs from c(tk−1) per unit time. When the split rate is low, c(tk) is larger than other split communities. Otherwise, c(tk) is smaller than other split communities.

$$R_{split}(c(t_{k-1}),c(t_k)) = \frac{N_{sp}(c(t_{k-1}))}{t_k - t_{k-1}}$$

Other Metrics

The novelty metrics of a main line ($c(ti)$, $c(ti+1)$, ..., $c(t\,j)$) is calculated as follows.

$$R_{novelty}(c(t_i), c(t_j)) = \frac{\sum_{k=i}^{j} N_{ap}(c(t_k))}{t_j - t_i}$$

Web Archives and Graphs

- Web archiving is the process of collecting portions of the Web to ensure the information is preserved in an archive
- Web crawlers are used for automated capture due to the massive size and amount of information on the Web.
- From each archive, a Web graph is built with URLs and links by extracting anchors from all pages in the archive.
- The graph included not only URLs inside the archive, but also URLs outside pointed to by inside URLs.
- By comparing these graphs, the Web was extremely dynamic

The size distribution of communities also follows the power law and its exponent did not change so much over time. Although the size distribution of communities is stable, the structure of communities changes dynamically. The structure of the chart changes mainly by split and merge, in which more than half of communities are involved.

Split and Merged Communities

- Both distributions roughly follow the power law, and show that split or merge rate is small in most cases.
- Their shapes and scales are also similar.

- This symmetry is part of the reason why the size distribution of communities does not change so much.

Emerged and Dissolved Communities

- The size distributions of emerged and dissolved communities also follow the power law
- Contribute to preserve the size distribution of communities.
- Small communities are easy to emerge and dissolve

Growth Rate

- The growth rate is small for most of communities, and the graph has clear y-axis symmetry.
- Size distribution of communities is preserved over time.

Combining evolution metrics and relevance, evolution around a particular community can be located. The size distribution of communities followed the power-law, and its exponent did not change so much over time.

2.6 DETECTING COMMUNITIES IN SOCIAL NETWORKS

Detecting communities from given social networks are practically important for the following reasons:

1. Communities can be used for information recommendation because members of the communities often have similar tastes and preferences. Membership of detected communities will be the basis of collaborative filtering.

2. Communities will help us understand the structures of given social networks. Communities are regarded as components of given social networks, and they will clarify the functions and properties of the networks.

3. Communities will play important roles when we visualize large-scale social networks. Relations of the communities clarify the processes of information sharing and information diffusions, and they may give us some insights for the growth the networks in the future.

2.7 EVALUATING COMMUNITIES

It is necessary to establish which partition exhibit a real community structure. Therefore, a quality function for evaluating how good a partition is needed. The most popular quality function is the modularity of Newman and Girivan:

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

where the sum runs over all pairs of vertices, A is the adjacency matrix, ki is the degree of vertex i and m is the total number of edges of the network. Modularity can be rewritten as follows:

$$Q = \sum_{s=1}^{nm} \left[\frac{l_s}{m} - \left(\frac{d_s}{2m} \right)^2 \right]$$

where nm is the number of communities, ls is the total number of edges joining vertices of community s, and ds is the sum of the degrees of the vertices of s. The first term of each summand is the fraction of edges of the network inside the community, whereas the second term represents the expected fraction of edges that

would be there if the network were a random network with the same degree for each vertex. Figure 3.b illustrates the meaning of modularity.

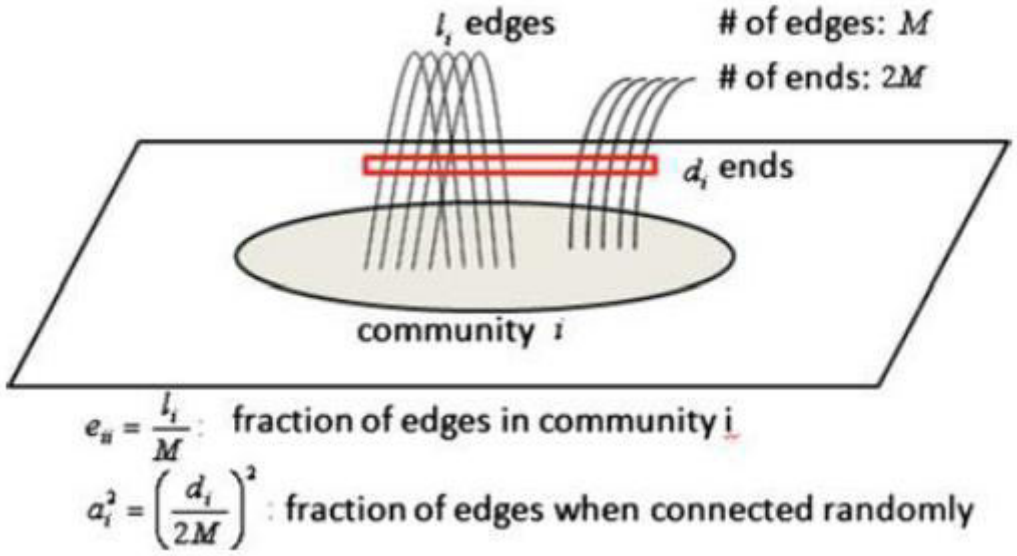


Fig.1 Modularity

The latter formula implicitly shows the definition of a community: a sub network is a community if the number of edges inside it is larger than the expected number in modularity's null model. The modularity of the whole network, taken as a single community, is zero. Modularity is always smaller than one, and it can be negative as well.

2.8 CORE METHODS FOR COMMUNITY DETECTION AND MINING

There are naive methods for dividing given networks into sub networks, such as graph partitioning, hierarchical clustering, and k-means clustering. The methods for detecting communities are roughly classified into the following categories:

(1) Divisive algorithms

(2) Modularity optimization

(3) Spectral algorithms and

(4) Other algorithms

Divisive Algorithms:

A simple way to identify communities in a network is to detect the edges that connect vertices of different communities and remove them, so that the communities get disconnected from each other. The steps of the algorithm are as follows:

(1) Computation of the centrality of all edges,

(2) Removal of edge with largest centrality,

(3) Recalculation of centralities on the running network, and

(4) Iteration of the cycle from step (2).

Edge betweenness is the number of shortest paths between all vertex pairs that run along the edge.

Modularity Optimization:

Modularity is a quality function for evaluating partitions. Therefore, the partition corresponding to its maximum value on a given network should be the best one. This is the main idea for modularity optimization. It has been proved that modularity optimization is an NPhard problem. However, there are currently several algorithms that are able to find fairly good approximations of the modularity maximum in a reasonable time. One of the famous algorithms for modularity optimization is CNM algorithm. Another example of the algorithms are greedy algorithms and simulated annealing.

Spectral Algorithms:

Spectral algorithms are to cut given network into pieces so that the number of edges to be cut will be minimized. One of the basic algorithms is spectral graph bipartitioning. The Laplacian matrix L of a network is an n * n symmetric matrix, with one row and column for each vertex. Laplacian matrix is defined as L =D - A , where A is the adjacency matrix and D is the diagonal degree matrix with

$$Dii = \sum_k A_{ik}$$

All eigenvalues of L are real and non-negative, and L has a full set of n real and orthogonal eigenvectors. In order to minimize the above cut, vertices are partitioned based on the signs of the eigenvector that corresponds to the second smallest eigenvalue of L. In general, community detection based on repetitive bipartitioning is relatively fast.

Other Algorithms:

There are many other algorithms for detecting communities, such as the methods focusing on random walk, and the ones searching for overlapping cliques.

2.9 APPLICATIONS OF COMMUNITY MINING ALGORITHMS

Some applications of community mining, with respect to various tasks in social network analysis are listed below:

Network Reduction:

Network reduction is an important step in analyzing social networks. The example discussed here is taken from the work in which the network was

constructed from the bibliography of the book entitled “graph products: structure and recognition”. The bibliography contains 360 papers written by 314 authors.

Its corresponding network is a bipartite graph, in which each node denotes either one author or one paper, and link (i, j) represents author i publishing a paper j. Community structure is detected using a community mining algorithm called ICS. Each community contains some papers and their corresponding coauthors.

Most of the detected communities are self-connected components. Moreover, the clustered coauthor network can be reduced into a much smaller one by condensing each community as one node. Finally, the top-level condensed network corresponding to a 3-community structure is constructed by using ICS from the condensed network. From this a dendrogram corresponding to the original coauthor network can be built.

Discovering Scientific Collaboration Groups from Social Networks

This section show how community mining techniques can be applied to the analysis of scientific collaborations among researchers. Flink is a social network that describes the scientific collaborations among 681 semantic Web researchers (http://flink.semanticweb.org/).

The network was constructed based on semantic Web technologies and all related semantic information was automatically extracted from “Web-accessible information sources”, such as “Web pages, FOAF profiles, email lists, and publication archives”. The weights on the links measure the degrees of collaboration.

Mining Communities from Distributed and Dynamic Networks:

Many applications involve distributed and dynamically-evolving networks, in which resources and controls are not only decentralized but also updated frequently. One promising solution is based on an Autonomy-Oriented Computing (AOC) approach, in which a group of self-organizing agents are utilized. The agents will rely only on their locally acquired information about networks. Intelligent Portable Digital Assistants (or iPDAs for short) that people carry around can form a distributed network, in which their users communicate with each other through calls or messages.

One useful function of iPDAs would be to find and recommend new friends with common interests, or potential partners in research or business, to the users. The way to implement it will be through the following steps:

(1) Based on an iPDA user's communication traces, selecting individuals who have frequently contacted or been contacted with the user during a certain period of time;

(2) Taking the selected individuals as the input to an AOC-based algorithm.

(3) Ranking and recommending new persons who might not be included the current acquaintance book, the user.

In such a way, people can periodically receive recommendations about friends or partners from their iPDAs.

SOCIAL NETWORK SOFTWARE

SNA software generates features from raw network data formatted in an edgelist, adjacency list, or adjacency matrix (also called sociomatrix), often combined with (individual/node-level) attribute data. Though the majority of network analysis software uses a plain text ASCII data format, some software packages contain the capability to utilize relational databases to import and/or store network features. visual representations of social networks are important to

understand network data and convey the result of the analysis. Visualization often also facilitates qualitative interpretation of network data. With respect to visualization, network analysis tools are used to change the layout, colors, size and other properties of the network representation.

Some SNA software can perform predictive analysis. This includes using network phenomena such as a tie to predict individual level outcomes (often called peer influence or contagion modeling), using individual-level phenomena to predict network outcomes such as the formation of a tie/edge (often called homophily models) or particular type of triad, or using network phenomena to predict other network phenomena, such as using a triad formation at time 0 to predict tie formation at time 1.

PRODUCT	MAIN FUNCTIONALITY	PLATFORM	LICENSE AND COST	NOTES
Allegrograph	Graph Database. RDF with Gruff visualization tool	Linux, Mac, Windows	Free and Commercial	AllegroGraph is a graph database. It is disk-based, fully transactional OLTP database that stores data structured in graphs rather than in tables. AllegroGraph includes a Social Networking Analytics library.
Gephi	Graph exploration and manipulation software	Any system supporting Java 1.6 and OpenGL	Open Source (GPL3), seeking contributors	Gephi[1] is an interactive visualization and exploration platform for all kinds of networks and complex systems, dynamic and hierarchical graphs. It is a tool for people that have to explore and understand graphs. The user interacts with the representation.
Graph Stream	Dynamic Graph Library	Any system supporting Java	Open Source	With GraphStream you deal with graphs. Static and Dynamic. You create them from scratch, from a file or any source. You display and render them.

Java Universal Network/Graph (JUNG) Framework	network and graph manipulation, analysis, and visualization	Any platform supporting Java	Open source (BSD license)	JUNG is a Java API and library that provides a common and extensible language for the modeling, analysis, and visualization of relational data. It supports a variety of graph types (including hypergraphs), supports graph elements of any type and with any properties
Mathematica	Graph analysis, statistics, data visualization, optimization, image recognitio	Windows, Macintosh, Linux	Commercial	Mathematica is a general purpose computation and analysis environment.