STATICAL ANALYSIS OF SOCIAL NETWORKS

Enrolment No.	:101322
Name of Student	:Shreya Tyagi
Name of supervisor(s)	:Dr Nitin Chanderwal

May – 2014

Submitted in partial fulfillment of the Degree of
Bachelor of Technology

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY,
WAKNAGHAT

TABLE OF CONTENTS

CERTIFICATE

This is to certify that the work titled “**Statical Analysis of Social Networks**” submitted by “**Shreya Tyagi**” in partial fulfillment for the award of degree of “**BTech in Computer Science & Engineering**” of Jaypee University of Information Technology, Waknaghat has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature of Supervisor ……………………..

Name of Supervisor ……………………..

Designation ……………………..

Date ……………………..

ACKNOWLEDGEMENTS

I acknowledge with deep regard the positive feedback and perpetual support offered by our Director of Student Affairs, Brigadier Balbir Singh during the initiation and development of this project. I would like to express my deepest gratitude to my project guide and supervisor, Dr Nitin Chanderwal for his constant support, encouragement and brilliant inputs, and his deep, abiding faith in my ability to complete this project successfully. I would also like to acknowledge the contributions of all the other members of the faculty at Jaypee University of Information Technology, Waknaghat, for contributing towards my academic pursuit.

Signature of the student ……………………..
Name of Student ……………………..
Date ……………………..

SUMMARY

Social network analysis is a new research field in data mining. The clustering in social network analysis is different from traditional clustering. It requires grouping objects into classes based on their links as well as their attributes. While traditional clustering algorithms group objects only based on objects' similarity, and it can't be applied to social network analysis. So on the basis of BSP (business system planning) clustering algorithm, a social network clustering analysis algorithm is proposed. The proposed algorithm, different from traditional BSP clustering algorithms, can group objects in a social network into different classes based on their links and identify relation among classes dynamically & require less amount of memory .

___________________	___________________
Signature of Student	Signature of Supervisor
Name: Shreya Tyagi	Name: Dr Nitin
Date:	Date:

LIST OF FIGURES

INTRODUCTION

A social network is a social structure made up of individuals (or organizations) called "nodes", which are tied (connected) by one or more specific types of interdependency, such as friendship, kinship, common interest, financial exchange, dislike, sexual relationships, or relationships of beliefs, knowledge or prestige.

Social network analysis views social relationships in terms of network theory consisting of nodes and ties (also called edges, links, or connections). Nodes are the individual actors within the networks, and ties are the relationships between the actors. The resulting graph-based structures are often very complex. There can be many kinds of ties between the nodes. Research in a number of academic fields has shown that social networks operate on many levels, from families up to the level of nations, and play a critical role in determining the way problems are solved, organizations are run, and the degree to which individuals succeed in achieving their goals.

In its simplest form, a social network is a map of specified ties, such as friendship, between the nodes being studied. The nodes to which an individual is thus connected are the social contacts of that individual. The network can also be used to measure social capital – the value that an individual gets from the social network. These concepts are often displayed in a social network diagram, where nodes are the points and ties are the lines.

Social network analysis

Social network analysis (related to network theory) has emerged as a key technique in modern sociology. It has also gained a significant following in anthropology, biology, communication studies, economics, geography, information science, organizational studies, social psychology, and sociolinguistics, and has become a popular topic of speculation and study. People have used the idea of "social network" loosely for over a century to connote complex sets of relationships between members of social systems at all scales, from interpersonal to international.

In 1954, J. A. Barnes started using the term systematically to denote patterns of ties, encompassing concepts traditionally used by the public and those used by social scientists: bounded groups (e.g., tribes, families) and social categories (e.g., gender, ethnicity). Scholars such as S.D. Berkowitz, Stephen Borgatti, Ronald Burt, Kathleen Carley, Martin Everett, Katherine Faust, Linton Freeman, Mark Granovetter, David Knoke, David Krackhardt, Peter Marsden, Nicholas Mullins, Anatol Rapoport, Stanley Wasserman, Barry Wellman, Douglas R. White, and Harrison White expanded the use of systematic social network analysis.

Social network analysis has now moved from being a suggestive metaphor to an analytic approach to a paradigm, with its own theoretical statements, methods, social network analysis software, and researchers. Analysts reason from whole to part; from structure to relation to individual; from behavior to attitude. They typically either study whole networks (also known as complete networks), all of the ties containing specified relations in a defined population, or personal networks (also known as egocentric networks), the ties that specified people have, such as their "personal communities". The distinction between whole/complete networks and personal/egocentric networks has depended largely on how analysts were able to gather data.

That is, for groups such as companies, schools, or membership societies, the analyst was expected to have complete information about who was in the network, all participants being both potential egos and alters. Personal/egocentric studies were typically conducted when identities of egos were known, but not their alters. These studies rely on the egos to provide information about the identities of alters and there is no expectation that the various egos or sets of alters will be tied to each other. A snowball network refers to the idea that the alters identified in an egocentric survey then become egos themselves and are able in turn to nominate additional alters. While there are severe logistic limits to conducting snowball network studies, a method for examining hybrid networks has recently been developed in which egos in complete networks can nominate alters otherwise not listed who are then available for all subsequent egos to see. The hybrid network may be valuable for examining whole/complete networks that are expected to include important players beyond those who are formally identified. For example, employees of a company often work with non-company consultants who may be part of a network that cannot fully be defined prior to data collection Several analytic tendencies distinguish social network analysis:
There is no assumption that groups are the building blocks of society: the approach is open to studying less-bounded social systems, from nonlocal communities to links among websites.
Rather than treating individuals (persons, organizations, states) as discrete units of analysis, it focuses on how the structure of ties affects individuals and their relationships.
In contrast to analyses that assume that socialization into norms determines behavior, network analysis looks to see the extent to which the structure and composition of ties affect norms.
The shape of a social network helps determine a network's usefulness to its individuals. Smaller, tighter networks can be less useful to their members than networks with lots of loose connections (weak ties) to individuals outside the main network. More open networks, with many weak ties and social connections, are more likely to introduce new ideas and opportunities to their members than closed networks with many redundant ties. In other words, a group of friends who only do things with each other already share the same knowledge and opportunities. A group of individuals with connections to other social worlds is likely to have access to a wider range of information. It is better for individual success to have connections to a variety of networks rather than many connections within a single network. Similarly, individuals can exercise influence or act as brokers within their social networks by bridging two networks that are not directly linked (called filling structural holes).
The power of social network analysis stems from its difference from traditional social scientific studies, which assume that it is the attributes of individual actors—whether they are friendly or unfriendly, smart or dumb, etc.—that matter. Social network analysis produces an alternate view, where the attributes of individuals are less important than their relationships and ties with other actors within the network. This approach has turned out to be useful for explaining many real-world phenomena, but leaves less room for individual agency, the ability for individuals to influence their success, because so much of it rests within the structure of their network.
Social networks have also been used to examine how organizations interact with each other, characterizing the many informal connections that link executives together, as well as associations and connections between individual employees at different organizations. For example, power within organizations often comes more from the degree to which an individual within a network is at the center of many relationships than actual job title. Social networks also play a key role in hiring, in business success, and in job performance. Networks provide ways for companies to gather information, deter competition, and collude in setting prices or policies.

History of social network analysis

A summary of the progress of social networks and social network analysis has been written by Linton Freeman. Precursors of social networks in the late 1800s include Émile Durkheim and Ferdinand Tönnies. Tönnies argued that social groups can exist as personal and direct social ties that either link individuals who share values and belief (gemeinschaft) or impersonal, formal, and instrumental social links (gesellschaft). Durkheim gave a non-individualistic explanation of social facts arguing that social phenomena arise when interacting individuals constitute a reality that can no longer be accounted for in terms of the properties of individual actors. He distinguished between a traditional society – "mechanical solidarity" – which prevails if individual differences are minimized, and the modern society – "organic solidarity" – that develops out of cooperation between differentiated individuals with independent roles.

Georg Simmel, writing at the turn of the twentieth century, was the first scholar to think directly in social network terms. His essays pointed to the nature of network size on interaction and to the likelihood of interaction in ramified, loosely-knit networks rather than groups (Simmel, 1908/1971).

After a hiatus in the first decades of the twentieth century, three main traditions in social networks appeared. In the 1930s, J.L. Moreno pioneered the systematic recording and analysis of social interaction in small groups, especially classrooms and work groups (sociometry), while a Harvard group led by W. Lloyd Warner and Elton Mayo explored interpersonal relations at work. In 1940, A.R. Radcliffe-Brown's presidential address to British anthropologists urged the systematic study of networks. However, it took about 15 years before this call was followed-up systematically.

Social network analysis developed with the kinship studies of Elizabeth Bott in England in the 1950s and the 1950s–1960s urbanization studies of the University of Manchester group of anthropologists (centered around Max Gluckman and later J. Clyde Mitchell) investigating community networks in southern Africa, India and the United Kingdom. Concomitantly, British anthropologist S.F. Nadel codified a theory of social structure that was influential in later network analysis.

In the 1960s-1970s, a growing number of scholars worked to combine the different tracks and traditions. One group was centered around Harrison White and his students at the Harvard University Department of Social Relations: Ivan Chase, Bonnie Erickson, Harriet Friedmann, Mark Granovetter, Nancy Howell, Joel Levine, Nicholas Mullins, John Padgett, Michael Schwartz and Barry Wellman. Also independently active in the Harvard Social Relations department at the time were Charles Tilly, who focused on networks in political and community sociology and social movements, and Stanley Milgram, who developed the "six degrees of separation" thesis.

Mark Granovetter and Barry Wellman are among the former students of White who have elaborated and popularized social network analysis. Significant independent work was also done by scholars elsewhere: University of California Irvine social scientists interested in mathematical applications, centered around Linton Freeman, including John Boyd, Susan Freeman, Kathryn Faust, A. Kimball Romney and Douglas White; quantitative analysts at the University of Chicago, including Joseph Galaskiewicz, Wendy Griswold, Edward Laumann, Peter Marsden, Martina Morris, and John Padgett; and communication scholars at Michigan State University, including Nan Lin and Everett Rogers. A substantively-oriented University of Toronto sociology group developed in the 1970s, centered on former students of Harrison White: S.D. Berkowitz, Harriet Friedmann, Nancy Leslie Howard, Nancy Howell, Lorne Tepperman and Barry Wellman, and also including noted modeler and game theorist Anatol Rapoport.In terms of theory, it critiqued methodological individualism and group-based analyses, arguing that seeing the world as social networks offered more analytic leverage.

Research

Social network analysis has been used in epidemiology to help understand how patterns of human contact aid or inhibit the spread of diseases such as HIV in a population. The evolution of social networks can sometimes be modeled by the use of agent based models, providing insight into the interplay between communication rules, rumor spreading and social structure.

SNA may also be an effective tool for mass surveillance – for example the Total Information Awareness program was doing in-depth research on strategies to analyze social networks to determine whether or not U.S. citizens were political threats. Diffusion of innovations theory explores social networks and their role in influencing the spread of new ideas and practices. Change agents and opinion leaders often play major roles in spurring the adoption of innovations, although factors inherent to the innovations also play a role.

Robin Dunbar has suggested that the typical size of an egocentric network is constrained to about 150 members due to possible limits in the capacity of the human communication channel. The rule arises from cross-cultural studies in sociology and especially anthropology of the maximum size of a village (in modern parlance most reasonably understood as an ecovillage). It is theorized in evolutionary psychology that the number may be some kind of limit of average human ability to recognize members and track emotional facts about all members of a group. However, it may be due to economics and the need to track "free riders", as it may be easier in larger groups to take advantage of the benefits of living in a community without contributing to those benefits.

Mark Granovetter found in one study that more numerous weak ties can be important in seeking information and innovation. Cliques have a tendency to have more homogeneous opinions as well as share many common traits. This homophilic tendency was the reason for the members of the cliques to be attracted together in the first place. However, being similar, each member of the clique would also know more or less what the other members knew. To find new information or insights, members of the clique will have to look beyond the clique to its other friends and acquaintances. This is what Granovetter called "the strength of weak ties". Guanxi is a central concept in Chinese society (and other East Asian cultures) that can be summarized as the use of personal influence. Guanxi can be studied from a social network approach. The small world phenomenon is the hypothesis that the chain of social acquaintances required to connect one arbitrary person to another arbitrary person anywhere in the world is generally short. The concept gave rise to the famous phrase six degrees of separation after a 1967 small world experiment by psychologist Stanley Milgram. In Milgram's experiment, a sample of US individuals were asked to reach a particular target person by passing a message along a chain of acquaintances. The average length of successful chains turned out to be about five intermediaries or six separation steps (the majority of chains in that study actually failed to complete). The methods (and ethics as well) of Milgram's experiment was later questioned by an American scholar, and some further research to replicate Milgram's findings had found that the degrees of connection needed could be higher. Academic researchers continue to explore this phenomenon as Internet-based communication technology has supplemented the phone and postal systems available during the times of Milgram. A recent electronic small world experiment at Columbia University found that about five to seven degrees of separation are sufficient for connecting any two people through e-mail.

Collaboration graphs can be used to illustrate good and bad relationships between humans. A positive edge between two nodes denotes a positive relationship (friendship, alliance, dating) and a negative edge between two nodes denotes a negative relationship (hatred, anger). Signed social network graphs can be used to predict the future evolution of the graph. In signed social networks, there is the concept of "balanced" and "unbalanced" cycles. A balanced cycle is defined as a cycle where the product of all the signs are positive. Balanced graphs represent a group of people who are unlikely to change their opinions of the other people in the group. Unbalanced graphs represent a group of people who are very likely to change their opinions of the people in their group. For example, a group of 3 people (A, B, and C) where A and B have a positive relationship, B and C have a positive relationship, but C and A have a negative relationship is an unbalanced cycle. This group is very likely to morph into a balanced cycle, such as one where B only has a good relationship with A, and both A and B have a negative relationship with C. By using the concept of balances and unbalanced cycles, the evolution of signed social network graphs can be predicted.

One study has found that happiness tends to be correlated in social networks. When a person is happy, nearby friends have a 25 percent higher chance of being happy themselves. Furthermore, people at the center of a social network tend to become happier in the future than those at the periphery. Clusters of happy and unhappy people were discerned within the studied networks, with a reach of three degrees of separation: a person's happiness was associated with the level of happiness of their friends' friends' friends. (See also Emotional contagion.)

Some researchers have suggested that human social networks may have a genetic basis. Using a sample of twins from the National Longitudinal Study of Adolescent Health, they found that in-degree (the number of times a person is named as a friend), transitivity (the probability that two friends are friends with one another), and betweenness centrality (the number of paths in the network that pass through a given person) are all significantly heritable.

Existing models of network formation cannot account for this intrinsic node variation, so the researchers propose an alternative "Attract and Introduce" model that can explain heritability and many other features of human social networks.

Metrics (measures) in social network analysis

Betweenness

The extent to which a node lies between other nodes in the network. This measure takes into account the connectivity of the node's neighbors, giving a higher value for nodes which bridge clusters. The measure reflects the number of people who a person is connecting indirectly through their direct links.

Bridge

An edge is said to be a bridge if deleting it would cause its endpoints to lie in different components of a graph.

Centrality

This measure gives a rough indication of the social power of a node based on how well they "connect" the
network. "Betweenness", "Closeness", and "Degree" are all measures of centrality.

Centralization
The difference between the number of links for each node divided by maximum possible sum of differences. A centralized network will have many of its links dispersed around one or a few nodes, while a decentralized network is one in which there is little variation between the number of links each node possesses.

Closeness
The degree an individual is near all other individuals in a network (directly or indirectly). It reflects the ability to access information through the "grapevine" of network members. Thus, closeness is the inverse of the sum of the shortest distances between each individual and every other person in the network. (See also: Proxemics)
The shortest path may also be known as the "geodesic distance".

Clustering coefficient
A measure of the likelihood that two associates of a node are associates themselves. A higher clustering coefficient indicates a greater 'cliquishness'.

Cohesion
The degree to which actors are connected directly to each other by cohesive bonds. Groups are identified as ‘cliques’ if every individual is directly tied to every other individual, ‘social circles’ if there is less stringency of direct contact, which is imprecise, or as structurally cohesive blocks if precision is wanted.[20]

Degree
The count of the number of ties to other actors in the network. See also degree (graph theory). (Individual-level)

Density
The degree a respondent's ties know one another/ proportion of ties among an individual's nominees. Network or global-level density is the proportion of ties in a network relative to the total number possible (sparse versus dense networks).

Flow betweenness centrality
The degree that a node contributes to sum of maximum flow between all pairs of nodes (not that node).

Eigenvector centrality
A measure of the importance of a node in a network. It assigns relative scores to all nodes in the network based on the principle that connections to nodes having a high score contribute more to the score of the node in question.

Local bridge
An edge is a local bridge if its endpoints share no common neighbors. Unlike a bridge, a local bridge is contained in a cycle.

Path length
The distances between pairs of nodes in the network. Average path-length is the average of these distances between all pairs of nodes.

Prestige
In a directed graph prestige is the term used to describe a node's centrality. "Degree Prestige", "Proximity Prestige", and "Status Prestige" are all measures of Prestige. See also degree (graph theory).

Radiality
Degree an individual's network reaches out into the network and provides novel information and influence.

Reach
The degree any member of a network can reach other members of the network.

Structural cohesion
The minimum number of members who, if removed from a group, would disconnect the group.[21]

Structural equivalence
Refers to the extent to which nodes have a common set of linkages to other nodes in the system. The nodes don't need to have any ties to each other to be structurally equivalent.

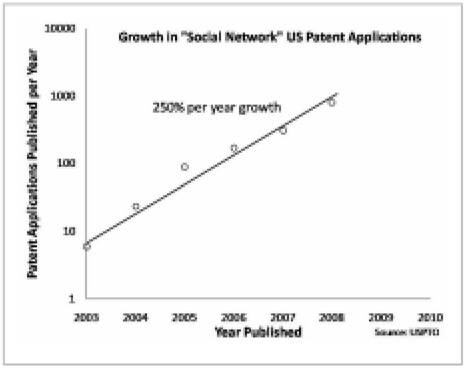
Structural hole
Static holes that can be strategically filled by connecting one or more links to link together other points. Linked to ideas of social capital: if you link to two people who are not linked you can control their communication.

Network analytic software
Network analytic tools are used to represent the nodes (agents) and edges (relationships) in a network, and to analyze the network data. Like other software tools, the data can be saved in external files. Additional information comparing the various data input formats used by network analysis software packages is available at NetWiki. Network analysis tools allow researchers to investigate large networks like the Internet, disease transmission, etc. These tools provide mathematical functions that can be applied to the network model.

Visualization of networks
Visual representation of social networks is important to understand the network data and convey the result of the analysis. Many of the analytic software have modules for network visualization. Exploration of the data is done through displaying nodes and ties in various layouts, and attributing colors, size and other advanced properties to nodes.

Typical representation of the network data are graphs in network layout (nodes and ties). These are not very easy-to-read and do not allow an intuitive interpretation. Various new methods have been developed in order to display network data in more intuitive format (e.g. Sociomapping). Especially when using social network analysis as a tool for facilitating change, different approaches of participatory network mapping have proven useful. Here participants / interviewers provide network data by actually mapping out the network (with pen and paper or digitally) during the data collection session. One benefit of this approach is that it allows researchers to collect qualitative data and ask clarifying questions while the network data is collected. Examples of network mapping techniques are Net-Map (pen-and-paper based) and VennMaker (digital)

Patents



There has been rapid growth in the number of US patent applications
that cover new technologies related to social networking. The number
of published applications has been growing at about 250% per year
over the past five years. There are now over 2000 published
applications. Only about 100 of these applications have issued as
patents, however, largely due to the multi-year backlog in examination
of business method patents and ethical issues connected with this
patent category

Analysis Software

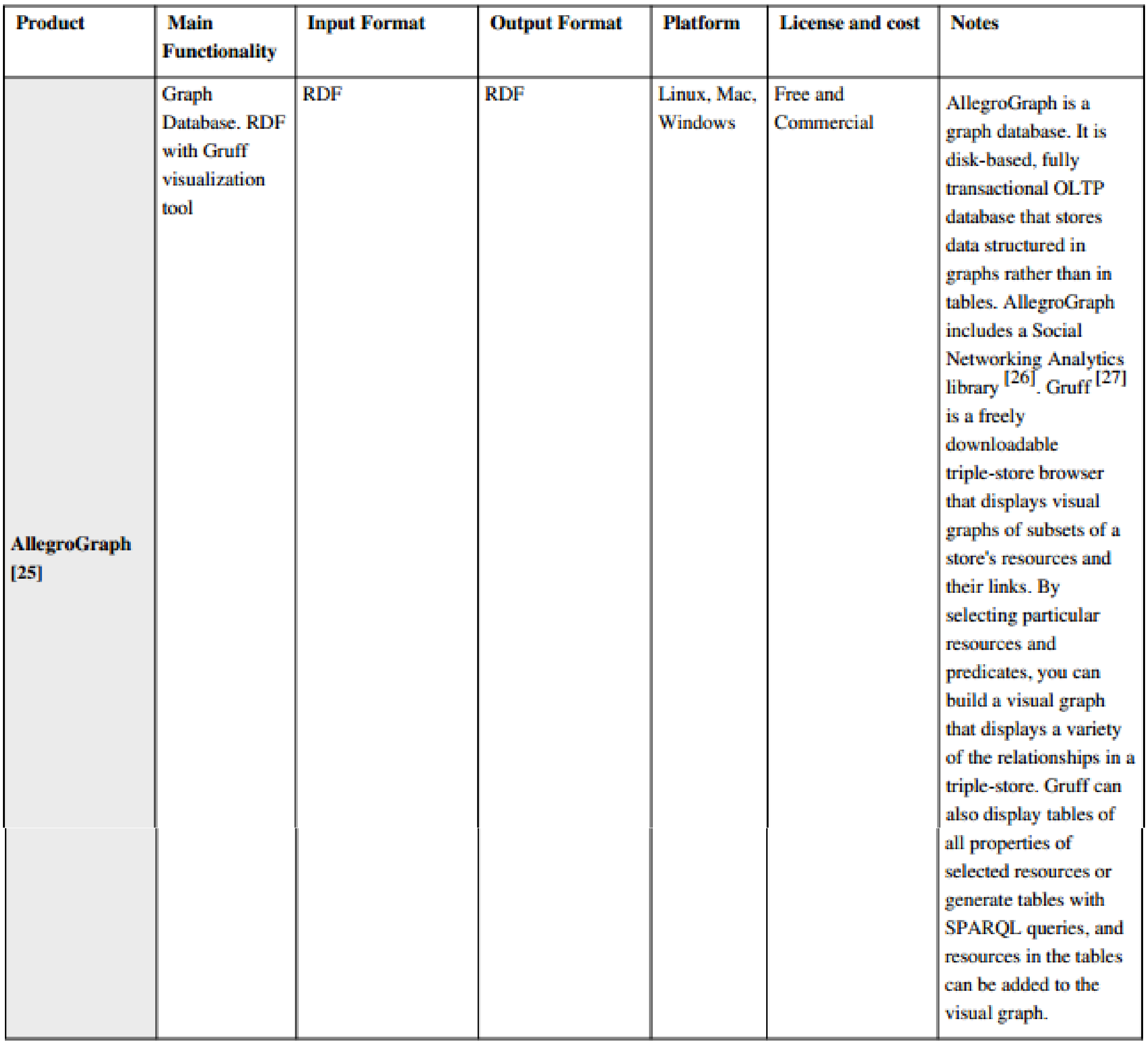
Social network analysis software

Social network analysis software is used to identify, represent, analyze, visualize, or simulate nodes (e.g. agents, organizations, or knowledge) and edges (relationships) from various types of input data (relational and non-relational), including mathematical models of social networks. The output data can be saved in external files.Various input and output file formats exist. Network analysis tools allow researchers to investigate representations of networks of different size - from small (e.g. families, project teams) to very large (e.g. the Internet, disease transmission). The various tools provide mathematical and statistical routines that can be applied to the network model.

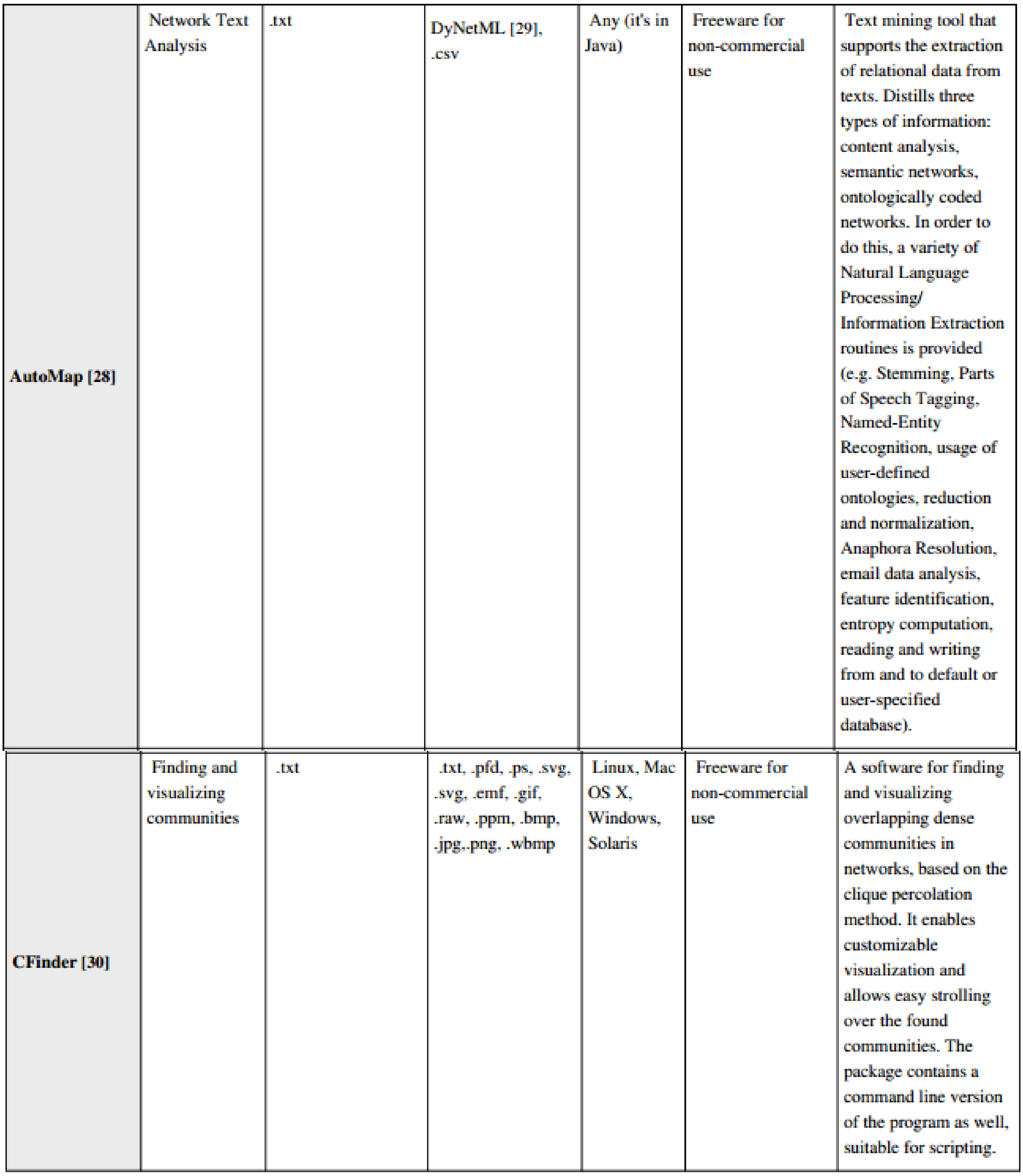
Visual representations of social networks are important to understand network data and convey the result of the analysis. Visualization is often used as an additional or standalone data analysis method. With respect to visualization, network analysis tools are used to change the layout, colors, size and other properties of the network representation.

Social network tools are:

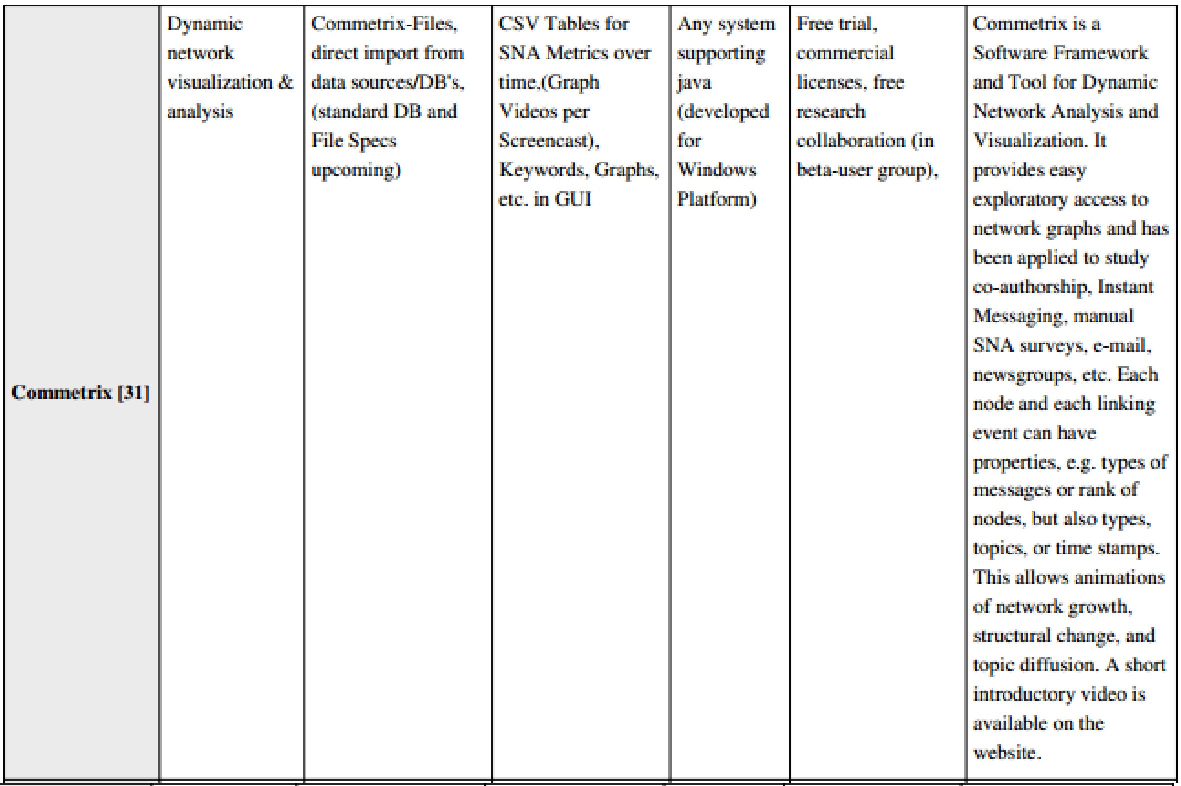
- For scholarly research tools like UCINet , Pajek , ORA, the statnet suite of packages in R, and GUESS are popular.
- Examples of business oriented social network tools include iPoint , NetMiner , InFlow, Keyhubs, Sentinel Visualizer, KXEN Social Network, NodeXL. For large networks with millions of nodes, try Sonamine or ORA. For mobile telecoms Idiro SNA Plus is recommended
- An open source package with GUI for Linux, Windows and Mac, is Social Networks Visualizer or SocNetV ,developed in Qt/C++.
- Another generic open source package for Windows, Linux and OS X with interfaces to Python and R is "igraph"
- Another generic open source package with [GUI] for Windows, Linux and OS X is "Tulip"
- RapidNet is a generic freely available open source solution for network analysis and interactive visual network exploration and drill-down.
- For Mac OS X a related package installer of SocNetV is available.
- For integrated egocentric data collection and visualization A systematic overview and comparison of a selection of software packages for social network analysis was provided by Huisman and Van Duijn. The International Network for Social Network Analysis (INSNA) maintains a large list of software packages and libraries.



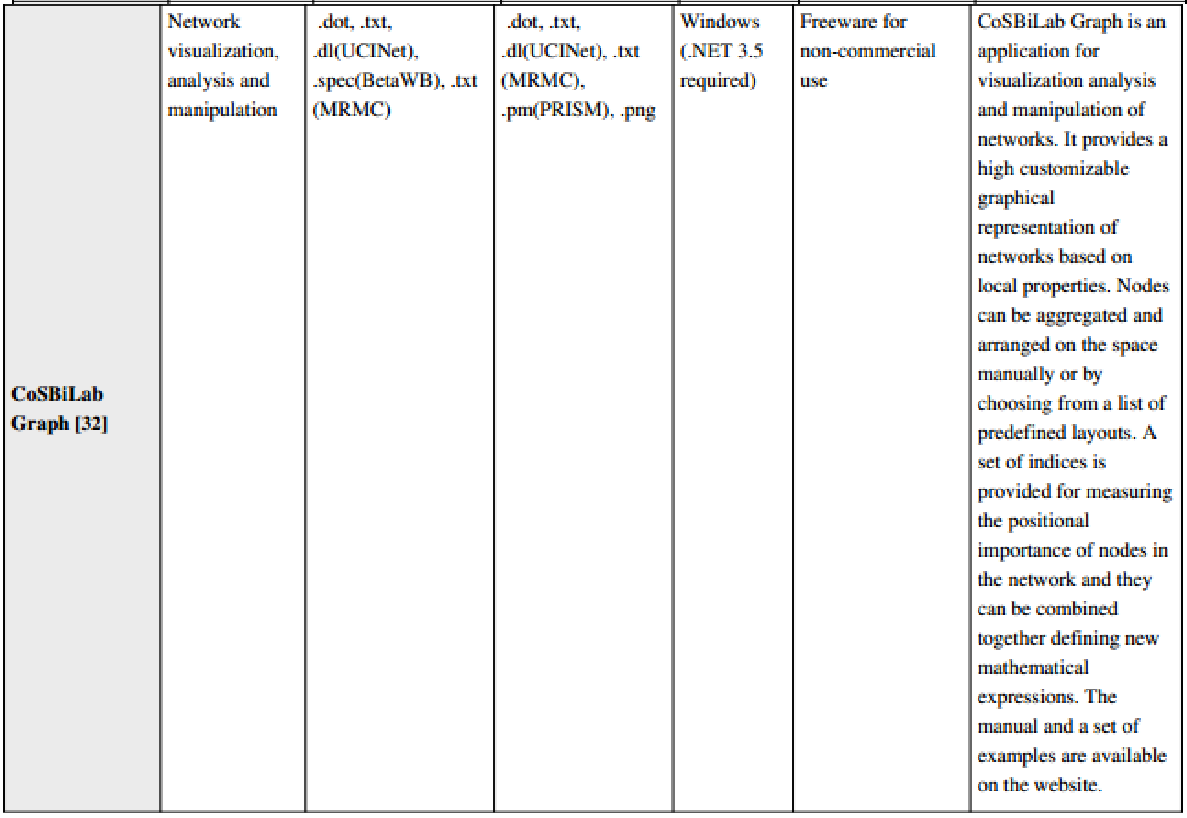
Product	Main Functionality	Input Format	Output Format	Platform	License and cost	Notes
AllegroGraph [25]	Graph Database. RDF with Gruff visualization tool	RDF	RDF	Linux, Mac, Windows	Free and Commercial	AllegroGraph is a graph database. It is disk-based, fully transactional OLTP database that stores data structured in graphs rather than in tables. AllegroGraph includes a Social Networking Analytics library [26]. Gruff [27] is a freely downloadable triple-store browser that displays visual graphs of subsets of a store's resources and their links. By selecting particular resources and predicates, you can build a visual graph that displays a variety of the relationships in a triple-store. Gruff can also display tables of all properties of selected resources or generate tables with SPARQL queries, and resources in the tables can be added to the visual graph.



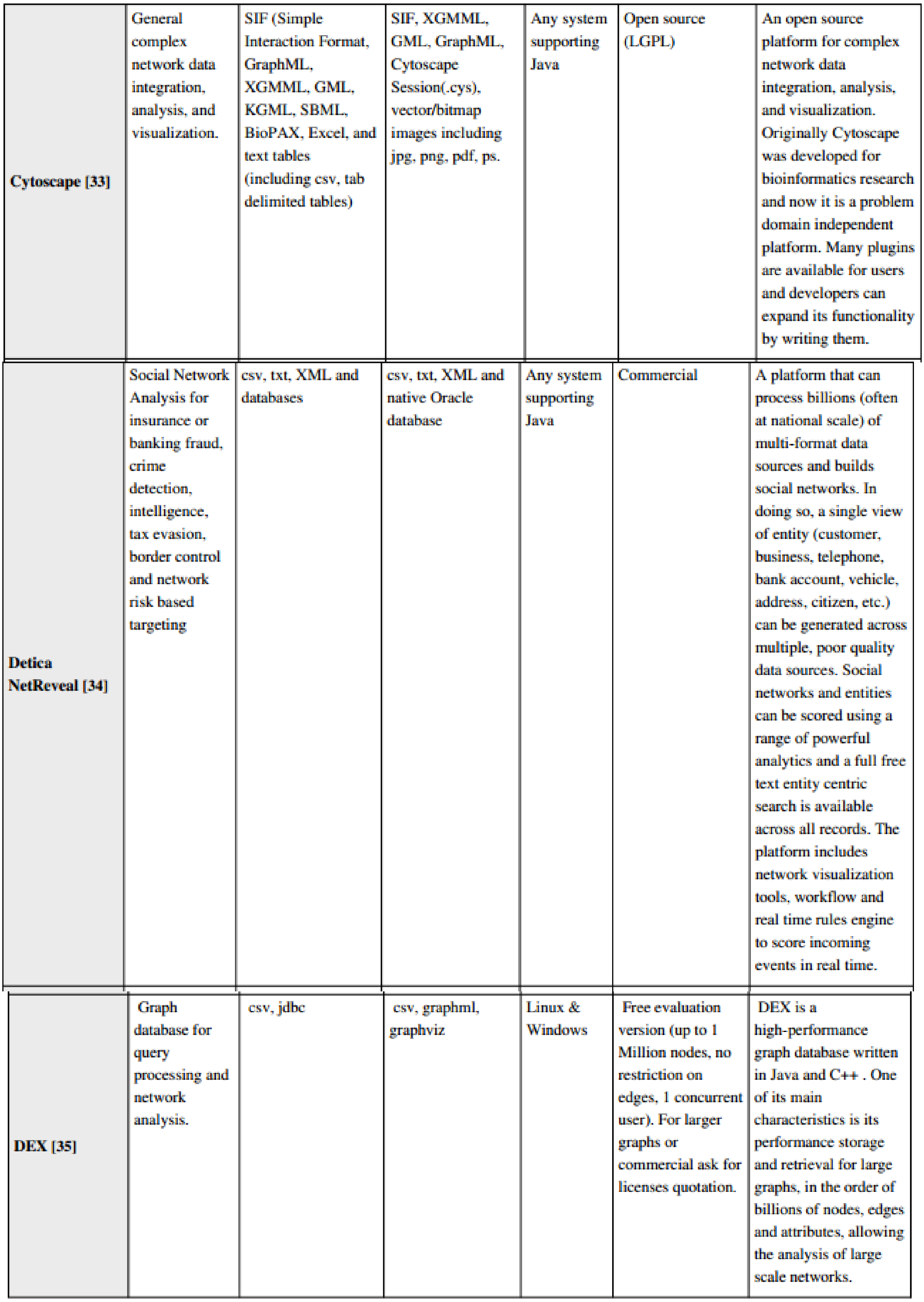
AutoMap [28]	Network Text Analysis	.txt	DyNetML [29], .csv	Any (it's in Java)	Freeware for non-commercial use	Text mining tool that supports the extraction of relational data from texts. Distills three types of information: content analysis, semantic networks, ontologically coded networks. In order to do this, a variety of Natural Language Processing/ Information Extraction routines is provided (e.g. Stemming, Parts of Speech Tagging, Named-Entity Recognition, usage of user-defined ontologies, reduction and normalization, Anaphora Resolution, email data analysis, feature identification, entropy computation, reading and writing from and to default or user-specified database).
CFinder [30]	Finding and visualizing communities	.txt	.txt, .pfd, .ps, .svg, .svg, .emf, .gif, .raw, .ppm, .bmp, .jpg,.png, .wbmp	Linux, Mac OS X, Windows, Solaris	Freeware for non-commercial use	A software for finding and visualizing overlapping dense communities in networks, based on the clique percolation method. It enables customizable visualization and allows easy strolling over the found communities. The package contains a command line version of the program as well, suitable for scripting.



Commetrix [31]	Dynamic network visualization & analysis	Commetrix-Files, direct import from data sources/DB's, (standard DB and File Specs upcoming)	CSV Tables for SNA Metrics over time,(Graph Videos per Screencast), Keywords, Graphs, etc. in GUI	Any system supporting java (developed for Windows Platform)	Free trial, commercial licenses, free research collaboration (in beta-user group),	Commetrix is a Software Framework and Tool for Dynamic Network Analysis and Visualization. It provides easy exploratory access to network graphs and has been applied to study co-authorship, Instant Messaging, manual SNA surveys, e-mail, newsgroups, etc. Each node and each linking event can have properties, e.g. types of messages or rank of nodes, but also types, topics, or time stamps. This allows animations of network growth, structural change, and topic diffusion. A short introductory video is available on the website.



CoSBiLab Graph [32]	Network visualization, analysis and manipulation	.dot, .txt, .dl(UCINet), .spec(BetaWB), .txt (MRMC)	.dot, .txt, .dl(UCINet), .txt (MRMC), .pm(PRISM), .png	Windows (.NET 3.5 required)	Freeware for non-commercial use	CoSBiLab Graph is an application for visualization analysis and manipulation of networks. It provides a high customizable graphical representation of networks based on local properties. Nodes can be aggregated and arranged on the space manually or by choosing from a list of predefined layouts. A set of indices is provided for measuring the positional importance of nodes in the network and they can be combined together defining new mathematical expressions. The manual and a set of examples are available on the website.



Cytoscape [33]	General complex network data integration, analysis, and visualization.	SIF (Simple Interaction Format, GraphML, XGMML, GML, KGML, SBML, BioPAX, Excel, and text tables (including csv, tab delimited tables)	SIF, XGMML, GML, GraphML, Cytoscape Session(.cys), vector/bitmap images including jpg, png, pdf, ps.	Any system supporting Java	Open source (LGPL)	An open source platform for complex network data integration, analysis, and visualization. Originally Cytoscape was developed for bioinformatics research and now it is a problem domain independent platform. Many plugins are available for users and developers can expand its functionality by writing them.
Detica NetReveal [34]	Social Network Analysis for insurance or banking fraud, crime detection, intelligence, tax evasion, border control and network risk based targeting	csv, txt, XML and databases	csv, txt, XML and native Oracle database	Any system supporting Java	Commercial	A platform that can process billions (often at national scale) of multi-format data sources and builds social networks. In doing so, a single view of entity (customer, business, telephone, bank account, vehicle, address, citizen, etc.) can be generated across multiple, poor quality data sources. Social networks and entities can be scored using a range of powerful analytics and a full free text entity centric search is available across all records. The platform includes network visualization tools, workflow and real time rules engine to score incoming events in real time.
DEX [35]	Graph database for query processing and network analysis.	csv, jdbc	csv, graphml, graphviz	Linux & Windows	Free evaluation version (up to 1 Million nodes, no restriction on edges, 1 concurrent user). For larger graphs or commercial ask for licenses quotation.	DEX is a high-performance graph database written in Java and C++ . One of its main characteristics is its performance storage and retrieval for large graphs, in the order of billions of nodes, edges and attributes, allowing the analysis of large scale networks.

Some Definitions

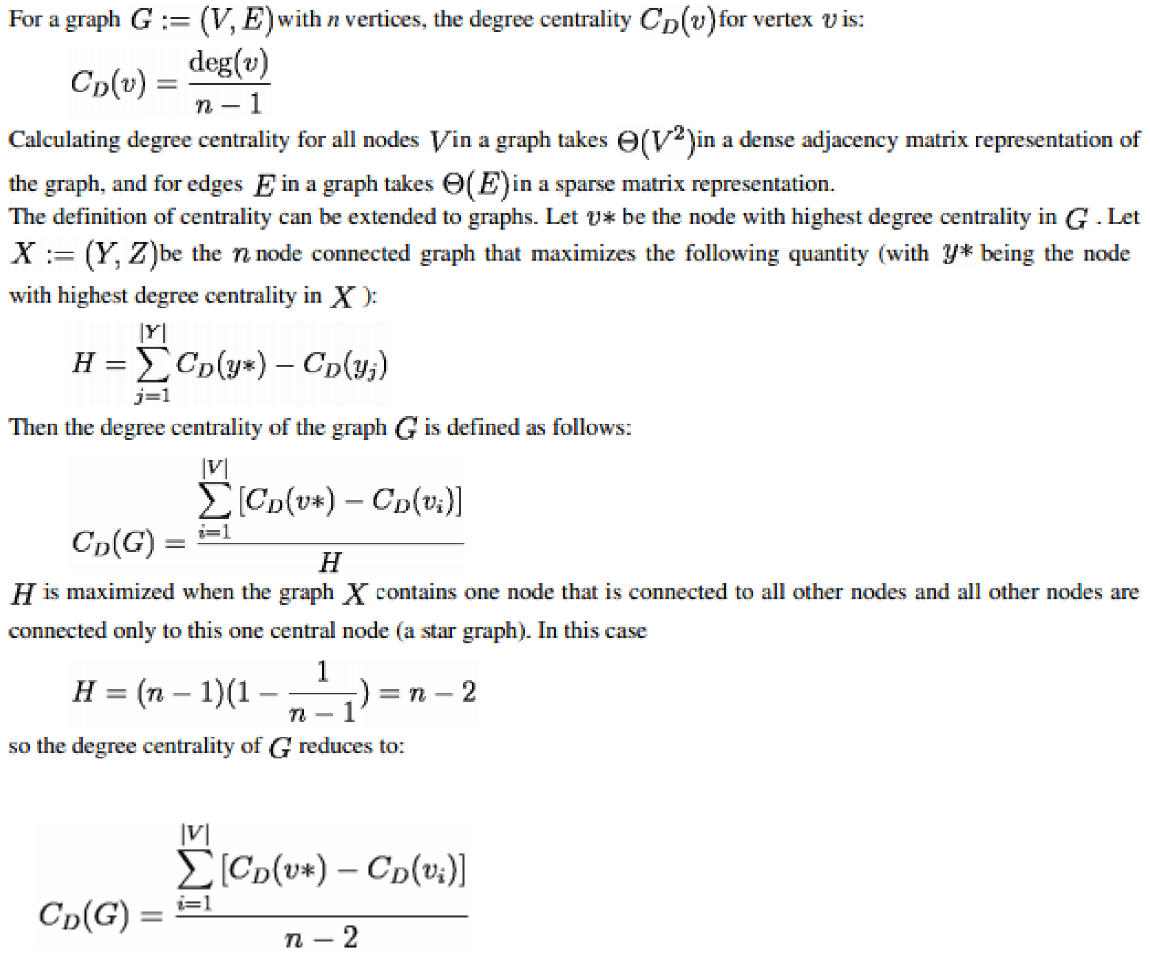
Betweenness

Within graph theory and network analysis, there are various measures of the centrality of a vertex within a graph that determine the relative importance of a vertex within the graph (for example, how important a person is within a social network, or, in the theory of space syntax, how important a room is within a building or how well-used a road is within an urban network).

There are four measures of centrality that are widely used in network analysis: degree centrality, betweenness, closeness, and eigenvector centrality. For a review as well as generalizations to weighted networks, see Opsahl et al.(2010).

Degree centrality

The first, and simplest, is degree centrality. Degree centrality is defined as the number of links incident upon a node (i.e., the number of ties that a node has). Degree is often interpreted in terms of the immediate risk of node for catching whatever is flowing through the network (such as a virus, or some information). If the network is directed (meaning that ties have direction), then we usually define two separate measures of degree centrality, namely indegree and outdegree. Indegree is a count of the number of ties directed to the node, and outdegree is the number of ties that the node directs to others. For positive relations such as friendship or advice, we normally interpret indegree as a form of popularity, and outdegree as gregariousness.



For a graph $G := (V, E)$ with n vertices, the degree centrality $C_D(v)$ for vertex v is:

$$C_D(v) = \frac{\deg(v)}{n-1}$$

Calculating degree centrality for all nodes V in a graph takes $\Theta(V^2)$ in a dense adjacency matrix representation of the graph, and for edges E in a graph takes $\Theta(E)$ in a sparse matrix representation.

The definition of centrality can be extended to graphs. Let $v*$ be the node with highest degree centrality in G. Let $X := (Y, Z)$ be the n node connected graph that maximizes the following quantity (with $y*$ being the node with highest degree centrality in X):

$$H = \sum_{j=1}^{|Y|} C_D(y*) - C_D(y_j)$$

Then the degree centrality of the graph G is defined as follows:

$$C_D(G) = \frac{\sum_{i=1}^{|V|} [C_D(v*) - C_D(v_i)]}{H}$$

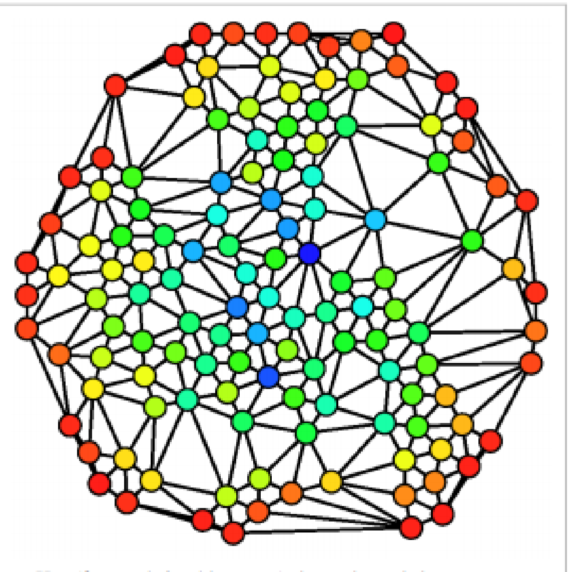
H is maximized when the graph X contains one node that is connected to all other nodes and all other nodes are connected only to this one central node (a star graph). In this case

$$H = (n-1)(1 - \frac{1}{n-1}) = n - 2$$

so the degree centrality of G reduces to:

$$C_D(G) = \frac{\sum_{i=1}^{|V|} [C_D(v*) - C_D(v_i)]}{n-2}$$

Betweenness centrality



Hue (from red=0 to blue=max) shows the node betweenness.

Betweenness is a centrality measure of a vertex within a graph (there is also edge betweenness, which is not discussed here). Vertices that occur on many shortest paths between other vertices have higher betweenness than those that do not.

For a graph $G := (V, E)$ with n vertices, the betweenness $C_B(v)$ for vertex v is computed as follows:

1. For each pair of vertices (s,t), compute all shortest paths between them.

2. For each pair of vertices (s,t), determine the fraction of shortest paths that pass through the vertex in question (here, vertex v).

3. Sum this fraction over all pairs of vertices (s,t).

Or, more succinctly:[2]

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} is the number of shortest paths from *s* to *t*, and $\sigma_{st}(v)$ is the number of shortest paths from *s* to *t* that pass through a vertex *v*. This may be normalised by dividing through the number of pairs of vertices not including *v*, which is $(n-1)(n-2)$ for directed graphs and $(n-1)(n-2)/2$ for undirected graphs. For example, in an undirected star graph, the center vertex (which is contained in every possible shortest path) would have a betweenness of $(n-1)(n-2)/2$ (1, if normalised) while the leaves (which are contained in no shortest paths) would have a betweenness of 0.

Calculating the betweenness and closeness centralities of all the vertices in a graph involves calculating the shortest paths between all pairs of vertices on a graph. This takes $\Theta(V^3)$ time with the Floyd–Warshall algorithm, modified to not only find one but count all shortest paths between two nodes. On a sparse graph, Johnson's algorithm may be more efficient, taking $O(V^2 \log V + VE)$ time. On unweighted graphs, calculating betweenness centrality takes $O(VE)$ time using Brandes' algorithm[3] .

In calculating betweenness and closeness centralities of all vertices in a graph, it is assumed that graphs are undirected and connected with the allowance of loops and multiple edges. When specifically dealing with network graphs, oftentimes graphs are without loops or multiple edges to maintain simple relationships (where edges represent connections between two people or vertices). In this case, using Brandes' algorithm will divide final centrality scores by 2 to account for each shortest path being counted twice[3] .

Closeness centrality

In topology and related areas in mathematics, closeness is one of the basic concepts in a topological space. Intuitively we say two sets are close if they are arbitrarily near to each other. The concept can be defined naturally in a metric space where a notion of distance between elements of the space is defined, but it can be generalized to topological spaces where we have no concrete way to measure distances.

In graph theory closeness is a centrality measure of a vertex within a graph. Vertices that are 'shallow' to other vertices (that is, those that tend to have short geodesic distances to other vertices within the graph) have higher closeness. Closeness is preferred in network analysis to mean shortest-path length, as it gives higher values to more central vertices, and so is usually positively associated with other measures such as degree.

In the network theory, closeness is a sophisticated measure of centrality. It is defined as the mean geodesic distance (i.e., the shortest path) between a vertex v and all other vertices reachable from it:

$$\frac{\sum_{t \in V \setminus v} d_G(v,t)}{n-1}$$

Where $n \geq 2$ is the size of the network's 'connectivity component' V reachable from v. Closeness can be regarded as a measure of how long it will take information to spread from a given vertex to other reachable vertices in the network[4].

Some define closeness to be the reciprocal of this quantity, but either way the information communicated is the same (this time estimating the speed instead of the timespan). The closeness for a vertex is the reciprocal of the sum of geodesic distances to all other vertices of V[5]:

$$C_C(v) = \sum_{t \in V \setminus v} 2^{-d_G(v,t)}.$$

Different methods and algorithms can be introduced to measure closeness, like the random-walk centrality introduced by Noh and Rieger (2003) that is a measure of the speed with which randomly walking messages reach a vertex from elsewhere in the network—a sort of random-walk version of closeness centrality.

The information centrality of Stephenson and Zelen (1989) is another closeness measure, which bears some similarity to that of Noh and Rieger. In essence it measures the harmonic mean length of paths ending at a vertex i, which is smaller if i has many short paths connecting it to other vertices. Dangalchev (2006), in order to measure the network vulnerability, modifies the definition for closeness so it can be used for disconnected graphs and the total closeness is easier to calculate:

$$C_C(v) = \sum_{t \in V \setminus v} 2^{-d_G(v,t)}.$$

Eigenvector Centrality

Eigenvector centrality is a measure of the importance of a node in a network. It assigns relative scores to all nodes in the network based on the principle that connections to high-scoring nodes contribute more to the score of the node in question than equal connections to low-scoring nodes. Google's PageRank is a variant of the Eigenvector centrality measure.

Using the adjacency matrix to find eigenvector centrality

Let x_i denote the score of the *i*th node. Let $A_{i,j}$ be the adjacency matrix of the network. Hence $A_{i,j} = 1$if the *i*th node is adjacent to the *j*th node, and $A_{i,j} = 0$otherwise. More generally, the entries in *A* can be real numbers representing connection strengths, as in a stochastic matrix.

For the i^{th} node, let the centrality score be proportional to the sum of the scores of all nodes which are connected to it. Hence

$$x_i = \frac{1}{\lambda} \sum_{j \in M(i)} x_j = \frac{1}{\lambda} \sum_{j=1}^{N} A_{i,j} x_j$$

where $M(i)$is the set of nodes that are connected to the i^{th} node, *N* is the total number of nodes and λ is a constant. In vector notation this can be rewritten as

$\mathbf{x} = \frac{1}{\lambda} \mathbf{A}\mathbf{x}$, or as the eigenvector equation $\mathbf{A}\mathbf{x} = \lambda \mathbf{x}$

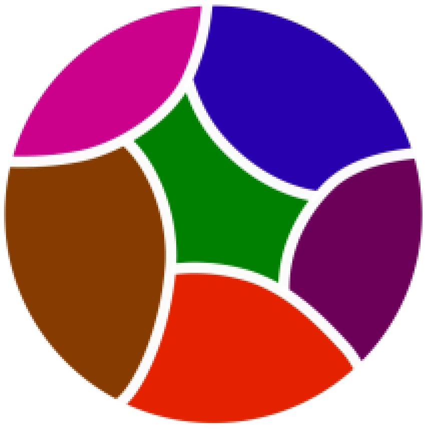
In general, there will be many different eigenvalues λ for which an eigenvector solution exists. However, the additional requirement that all the entries in the eigenvector be positive implies (by the Perron–Frobenius theorem) that only the greatest eigenvalue results in the desired centrality measure. The ith component of the related eigenvector then gives the centrality score of the ith node in the network. Power iteration is one of many eigenvalue algorithms that may be used to find this dominant eigenvector.

Equivalence relation

In mathematics, an equivalence relation is, loosely, a relation that specifies how to partition a set such that every element of the set is in exactly one of the blocks in the partition, and the union of all the blocks equals the original set. Two elements of the set are considered equivalent (with respect to the equivalence relation) if and only if they are elements of the same block.

Notation

Although various notations are used throughout the literature to denote that two elements a and b of a set are equivalent with respect to an equivalence relation R, the most common are "a ~ b" and "a ≡ b",which are used when R is the obvious relation being referenced, and variations of "a $\sim_R$ b", "a $\equiv_R$ b", or "aRb".



An equivalence relation partitions a set into several disjoint subsets, called equivalence classes. All the elements in a given equivalence class are equivalent among themselves, and no element is equivalent with any element from a different class.

Definition

A given binary relation ~ on a set A is said to be an equivalence relation if and only if it is reflexive, symmetric and transitive. Equivalently, for all a, b and c in A:

- a ~ a. (Reflexivity)
- if a ~ b then b ~ a. (Symmetry)
- if a ~ b and b ~ c then a ~ c. (Transitivity)

A together with the relation ~ is called a setoid. The equivalence class of a under ~, denoted [a], is defined as:

$$[a] = \{b \in A | a \sim b\} .$$

Reflexivity follows from symmetry and transitivity if for every element a∈A, there exists another element b∈A such that a~b holds. However, reflexivity does not follow from symmetry and transitivity alone. For example, let A be the set of integers, and let two elements of A be related if they are both even numbers. This relation is clearly symmetric and transitive, but in view of the existence of odd numbers, it is not reflexive.
On the other hand, let A be the set of integers, and let two elements of A be related if their difference is even. This is an equivalence relation, which partitions the integers into two equivalence classes, the even and odd integers.

Examples

Equivalence relations

The following are all equivalence relations:

- "Has the same birthday as" on the set of all people.
- "Is similar to" or "congruent to" on the set of all triangles.
- "Is congruent to modulo n" on the integers.
- "Has the same image under a function" on the elements of the domain of the function.
- "Is parallel to" on the set of subspaces of an affine space.

Relations that are not equivalences

- The relation "≥" between real numbers is reflexive and transitive, but not symmetric. For example, 7 ≥ 5 does not imply that 5 ≥ 7. It is, however, a partial order.
- The relation "has a common factor greater than 1 with" between natural numbers greater than 1, is reflexive and symmetric, but not transitive. (Example: The natural numbers 2 and 6 have a common factor greater than 1, and 6 and 3 have a common factor greater than 1, but 2 and 3 do not have a common factor greater than 1).
- The empty relation R on a non-empty set X (i.e. aRb is never true) is vacuously symmetric and transitive, but not reflexive. (If X is also empty then R is reflexive.)
- The relation "is approximately equal to" between real numbers, even if more precisely defined, is not an equivalence relation, because although reflexive and symmetric, it is not transitive, since multiple small changes can accumulate to become a big change. However, if the approximation is defined asymptotically, for example by saying that two functions f and g are approximately equal near some point if the limit of f-g is 0 at that point, then this defines an equivalence relation.
- The relation "is a sibling of" (used to connote pairs of distinct people who have the same parents) on the set of all human beings is not an equivalence relation. Although siblinghood is symmetric (if A is a sibling of B, then B is a sibling of A) and transitive on any 3 distinct people (if A is a sibling of B and C is a sibling of B, then A is a sibling of C, provided A is not C), it is not reflexive (A cannot be a sibling of A).

Connections to other relations

- A partial order is a relation that is reflexive, antisymmetric, and transitive.
- A congruence relation is an equivalence relation whose domain X is also the underlying set for an algebraic structure, and which respects the additional structure. In general, congruence relations play the role of kernels of homomorphisms, and the quotient of a structure by a congruence relation can be formed. In many important cases congruence relations have an alternative representation as substructures of the structure on which they are defined. E.g. the congruence relations on groups correspond to the normal subgroups.
- Equality is both an equivalence relation and a partial order. Equality is also the only relation on a set that is reflexive, symmetric and antisymmetric.
- A strict partial order is irreflexive, transitive, and asymmetric.
- A partial equivalence relation is transitive and symmetric. Transitive and symmetric imply reflexive if and only if for all a∈X exists b∈X such that a~b.
- A dependency relation is reflexive and symmetric.
- A preorder is reflexive and transitive.
- A compatibility relation is reflexive and symmetric.

Well-definedness under an equivalence relation

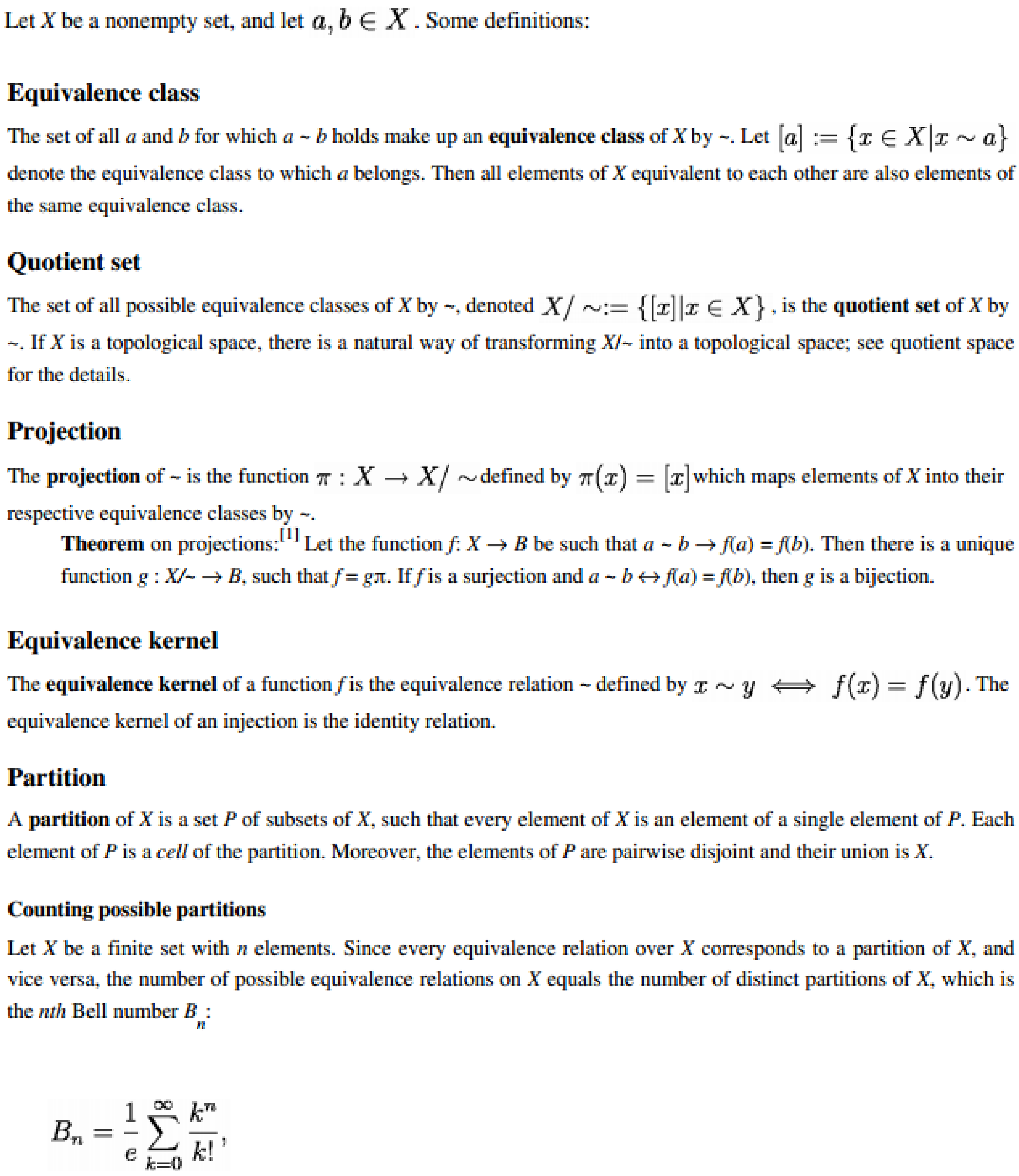
If ~ is an equivalence relation on X, and P(x) is a property of elements of X, such that whenever x ~ y, P(x) is true if P(y) is true, then the property P is said to be well-defined or a class invariant under the relation ~.

A frequent particular case occurs when f is a function from X to another set Y; if x 1 ~ x2 implies f(x1) = f(x2) then f is said to be a morphism for ~, a class invariant under ~, or simply invariant under ~. This occurs, e.g. in the character theory of finite groups. The latter case with the function f can be expressed by a commutative triangle. See also invariant. Some authors use "compatible with ~" or just "respects ~" instead of "invariant under ~".

More generally, a function may map equivalent arguments (under an equivalence relation ~A

) to equivalent values(under an equivalence relation ~B). Such a function is known as a morphism from ~A to ~B

Equivalence class, quotient set, partition



Let *X* be a nonempty set, and let $a, b \in X$. Some definitions:

Equivalence class

The set of all *a* and *b* for which *a* ~ *b* holds make up an **equivalence class** of *X* by ~. Let $[a] := \{x \in X | x \sim a\}$ denote the equivalence class to which *a* belongs. Then all elements of *X* equivalent to each other are also elements of the same equivalence class.

Quotient set

The set of all possible equivalence classes of *X* by ~, denoted $X/\sim := \{[x] | x \in X\}$, is the **quotient set** of *X* by ~. If *X* is a topological space, there is a natural way of transforming *X*/~ into a topological space; see quotient space for the details.

Projection

The **projection** of ~ is the function $\pi : X \to X/\sim$ defined by $\pi(x) = [x]$ which maps elements of *X* into their respective equivalence classes by ~.

> **Theorem** on projections:[1] Let the function *f*: *X* → *B* be such that *a* ~ *b* → *f*(*a*) = *f*(*b*). Then there is a unique function *g* : *X*/~ → *B*, such that *f* = *g*π. If *f* is a surjection and *a* ~ *b* ↔ *f*(*a*) = *f*(*b*), then *g* is a bijection.

Equivalence kernel

The **equivalence kernel** of a function *f* is the equivalence relation ~ defined by $x \sim y \iff f(x) = f(y)$. The equivalence kernel of an injection is the identity relation.

Partition

A **partition** of *X* is a set *P* of subsets of *X*, such that every element of *X* is an element of a single element of *P*. Each element of *P* is a *cell* of the partition. Moreover, the elements of *P* are pairwise disjoint and their union is *X*.

Counting possible partitions

Let *X* be a finite set with *n* elements. Since every equivalence relation over *X* corresponds to a partition of *X*, and vice versa, the number of possible equivalence relations on *X* equals the number of distinct partitions of *X*, which is the *nth* Bell number B_n:

$$B_n = \frac{1}{e} \sum_{k=0}^{\infty} \frac{k^n}{k!},$$

where the above is one of the ways to write the nth Bell number.

Fundamental Theorem of Equivalence Relations

A key result links equivalence relations and partitions:
- An equivalence relation ~ on a set X partitions X.
- Conversely, corresponding to any partition of X, there exists an equivalence relation ~ on X.

In both cases, the cells of the partition of X are the equivalence classes of X by ~. Since each element of X belongs to
a unique cell of any partition of X, and since each cell of the partition is identical to an equivalence class of X by ~,
each element of X belongs to a unique equivalence class of X by ~. Thus there is a natural bijection from the set of all
possible equivalence relations on X and the set of all partitions of X

Comparing equivalence relations
If ~ and ≈ are two equivalence relations on the same set S, and a~b implies a≈b for all a,b ∈ S, then ≈ is said to be a coarser relation than ~, and ~ is a finer relation than ≈. Equivalently,
- ~ is finer than ≈ if every equivalence class of ~ is a subset of an equivalence class of ≈, and thus every equivalence class of ≈ is a union of equivalence classes of ~.
- ~ is finer than ≈ if the partition created by ~ is a refinement of the partition created by ≈.

The equality equivalence relation is the finest equivalence relation on any set, while the trivial relation that makes all pairs of elements related is the coarsest.
The relation "~ is finer than ≈" on the collection of all equivalence relations on a fixed set is itself a partial order relation.
Generating equivalence relations
- Given any set X, there is an equivalence relation over the set [X→X] of all possible functions X→X. Two such functions are deemed equivalent when their respective sets of fixpoints have the same cardinality, corresponding to cycles of length one in a permutation. Functions equivalent in this manner form an equivalence class on
[X→X], and these equivalence classes partition [X→X].
- An equivalence relation ~ on X is the equivalence kernel of its surjective projection π : X → X/~.[4] Conversely, any surjection between sets determines a partition on its domain, the set of preimages of singletons in the codomain. Thus an equivalence relation over X, a partition of X, and a projection whose domain is X, are three equivalent ways of specifying the same thing.
- The intersection of any collection of equivalence relations over X (viewed as a subset of X × X) is also an equivalence relation. This yields a convenient way of generating an equivalence relation: given any binary relation R on X, the equivalence relation generated by R is the smallest equivalence relation containing R. Concretely, R generates the equivalence relation a ~ b if and only if there exist elements x1, x2, ..., xn in X such that a = x1, b = xn
, and (xi,xi+ 1)∈R or (xi+1,xi)∈R, i = 1, ..., n-1. Note that the equivalence relation generated in this manner can be trivial. For instance, the equivalence relation ~ generated by:
- Any total order on X has exactly one equivalence class, X itself, because x ~ y for all x and y;
- Any subset of the identity relation on X has equivalence classes that are the singletons of X.
- Equivalence relations can construct new spaces by "gluing things together." Let X be the unit Cartesian square [0,1] × [0,1], and let ~ be the equivalence relation on X defined by ∀a, b ∈ [0,1] ((a, 0) ~ (a, 1) ∧ (0, b) ~ (1, b)).

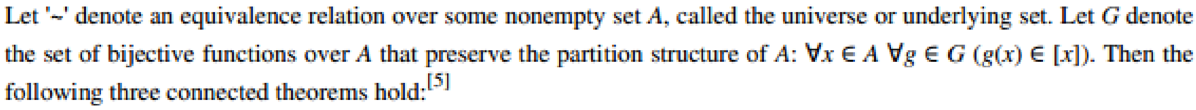
Then the quotient space X/~ can be naturally identified with a torus: take a square piece of paper, bend and glue together the upper and lower edge to form a cylinder, then bend the resulting cylinder so as to glue together its two open ends, resulting in a torus.

Algebraic structure

Much of mathematics is grounded in the study of equivalences, and order relations. It is very well known that lattice theory captures the mathematical structure of order relations. Even though equivalence relations are as ubiquitous in mathematics as order relations, the algebraic structure of equivalences is not as well known as that of orders. The former structure draws primarily on group theory and, to a lesser extent, on the theory of lattices, categories, and groupoids.

Group theory

Just as order relations are grounded in ordered sets, sets closed under pairwise supremum and infimum, equivalence relations are grounded in partitioned sets, which are sets closed under bijections and preserve partition structure. Since all such bijections map an equivalence class onto itself, such bijections are also known as permutations. Hence permutation groups (also known as transformation groups) and the related notion of orbit shed light on the mathematical structure of equivalence relations.



Let '~' denote an equivalence relation over some nonempty set *A*, called the universe or underlying set. Let *G* denote the set of bijective functions over *A* that preserve the partition structure of *A*: $\forall x \in A\ \forall g \in G\ (g(x) \in [x])$. Then the following three connected theorems hold:[5]

- ~ partitions A into equivalence classes. (This is the Fundamental Theorem of Equivalence Relations, mentioned above);
- Given a partition of A, G is a transformation group under composition, whose orbits are the cells of the partition‡;
- Given a transformation group G over A, there exists an equivalence relation ~ over A, whose equivalence classes are the orbits of G.

In sum, given an equivalence relation ~ over A, there exists a transformation group G over A whose orbits are the equivalence classes of A under ~. This transformation group characterisation of equivalence relations differs fundamentally from the way lattices characterize order relations. The arguments of the lattice theory operations meet and join are elements of some universe A. Meanwhile, the arguments of the transformation group operations composition and inverse are elements of a set of bijections, A → A.
Moving to groups in general, let H be a subgroup of some group G. Let ~ be an equivalence relation on G, such that a ~ b ↔ (ab−1 ∈ H). The equivalence classes of ~—also called the orbits of the action of H on G—are the right cosets of H in G. Interchanging a and b yields the left cosets.

‡Proof.[8] Let function composition interpret group multiplication, and function inverse interpret group inverse. Then G is a group under composition, meaning that ∀x ∈ A ∀g ∈ G ([g(x)] = [x]), because G satisfies the following four conditions:

- G is closed under composition. The composition of any two elements of G exists, because the domain and codomain of any element of G is A. Moreover, the composition of bijections is bijective;
- Existence of identity element. The identity function, I(x)=x, is an obvious element of G;
- Existence of inverse function. Every bijective function g has an inverse g−1
, such that gg−1 = I;
- Composition associates. f(gh) = (fg)h. This holds for all functions over all domains.

Let f and g be any two elements of G. By virtue of the definition of G, [g(f(x))] = [f(x)] and [f(x)] = [x], so that [g(f(x))] = [x]. Hence G is also a transformation group (and an automorphism group) because function composition preserves the partitioning of A.
Related thinking can be found in Rosen (2008: chpt. 10).

Categories and groupoids

The composition of morphisms central to category theory, denoted here by concatenation, generalizes the composition of functions central to transformation groups. The axioms of category theory assert that the composition of morphisms associates, and that the left and right identity morphisms exist for any morphism. If a morphism f has an inverse, f is an isomorphism, i.e., there exists a morphism g such that the compositions fg and gf equal the appropriate identity morphisms. Hence the category-theoretic concept nearest to an equivalence relation is a (small) category whose morphisms are all isomorphisms. Groupoid is another name for a small category of this nature.

Let G be a set and let "~" denote an equivalence relation over G. Then we can form a groupoid representing this equivalence relation as follows. The objects are the elements of G, and for any two elements x and y of G, there exists a unique morphism from x to y if and only if x~y. The elements x and y are "equivalent" if there is an element g of the groupoid from x to y. There may be many such g, each of which can be regarded as a distinct "proof" that x and y are equivalent.

The advantages of regarding an equivalence relation as a special case of a groupoid include:

- Whereas the notion of "free equivalence relation" does not exist, that of a free groupoid on a directed graph does. Thus it is meaningful to speak of a "presentation of an equivalence relation," i.e., a presentation of the corresponding groupoid;
- Bundles of groups, group actions, sets, and equivalence relations can be regarded as special cases of the notion of groupoid, a point of view that suggests a number of analogies;
- In many contexts "quotienting," and hence the appropriate equivalence relations often called congruences, are important. This leads to the notion of an internal groupoid in a category.

Lattices

The possible equivalence relations on any set X, when ordered by set inclusion, form a complete lattice, called Con X by convention. The canonical map ker: X^X → Con X, relates the monoid X^X of all functions on X and Con X. ker is surjective but not injective. Less formally, the equivalence relation ker on X, takes each function f: X→X to its kernel ker f. Likewise, ker(ker) is an equivalence relation on X^X. Equivalence relations and mathematical logic Equivalence relations are a ready source of examples or counterexamples. For example, an equivalence relation with exactly two infinite equivalence classes is an easy example of a theory which is ω-categorical, but not categorical for any larger cardinal number. An implication of model theory is that the properties defining a relation can be proved independent of each other (and hence necessary parts of the definition) if and only if, for each property, examples can be found of relations not satisfying the given property while satisfying all the other properties. Hence the three defining properties of equivalence relations can be proved mutually independent by the following three examples:

- Reflexive and transitive: The relation ≤ on N. Or any preorder;
- Symmetric and transitive: The relation R on N, defined as aRb ↔ ab ≠ 0. Or any partial equivalence relation;
- Reflexive and symmetric: The relation R on Z, defined as aRb ↔ "a − b is divisible by at least one of 2 or 3." Or any dependency relation.

Properties definable in first-order logic that an equivalence relation may or may not possess include:

- The number of equivalence classes is finite or infinite;
- The number of equivalence classes equals the (finite) natural number n;

- All equivalence classes have infinite cardinality;
- The number of elements in each equivalence class is the natural number n

Euclidean relations

Euclid's The Elements includes the following "Common Notion 1":
Things which equal the same thing also equal one another. Nowadays, the property described by Common Notion 1 is called Euclidean (replacing "equal" by "are in relation with"). The following theorem connects Euclidean relations and equivalence relations:

Theorem.
If a relation is Euclidean and reflexive, it is also symmetric and transitive.

Proof:
- (aRc ∧ bRc) → aRb [a/c] = (aRa ∧ bRa) → aRb [reflexive; erase T∧] = bRa → aRb. Hence R is symmetric.
- (aRc ∧ bRc) → aRb [symmetry] = (aRc ∧ cRb) → aRb. Hence R is transitive.

Hence an equivalence relation is a relation that is Euclidean and reflexive. The Elements mentions neither symmetry nor reflexivity, and Euclid probably would have deemed the reflexivity of equality too obvious to warrant explicit mention.

Centralization

Centralisation, or centralization (see spelling differences), is the process by which the activities of an organisation, particularly those regarding planning decision-making, become concentrated within a particular location and/or group. In political science, this refers to the concentration of a government's power - both geographically and politically, into a centralised government. In neuroscience, centralization refers to the evolutionary trend of the nervous system to be partitioned into a central nervous system and peripheral nervous system. In business studies centralisation and decentralisation is about where decisions are taken in the chain of command.

Clustering coefficient

In graph theory, a clustering coefficient is a measure of degree to which nodes in a graph tend to cluster together. Evidence suggests that in most real-world networks, and in particular social networks, nodes tend to create tightly knit groups characterised by a relatively high density of ties (Holland and Leinhardt, 1971P. W. Holland and S. Leinhardt (1998). "Transitivity in structural models of small groups". Comparative Group Studies 2: 107–124.; Watts and Strogatz, 1998D. J. Watts and Steven Strogatz (June 1998). "Collective dynamics of 'small-world' networks". Nature (journal)Nature 393 (6684): 440–442. doi:10.1038/30918. PMID 9623998. .). In real-world networks, this likelihood tends to be greater than the average probability of a tie randomly established between two nodes (Holland and Leinhardt, 1971; Watts and Strogatz, 1998).Two versions of this measure exist: the global and the local. The global version was designed to give an overall indication of the clustering in the network, whereas the local gives an indication of the embeddedness of single nodes. Global clustering coefficient The global clustering coefficient is based on triplets of nodes. A triplet is three nodes that are connected by either two (open triplet) or three (closed triplet) undirected ties. A triangle consists of three closed triplets, one centred on each of the nodes.

The global clustering coefficient is the number of closed triplets (or 3 x triangles) over the total number of triplets (both open and closed). The first attempt to measure it was made by Luce and Perry (1949)R. D. Luce and A. D. Perry (1949). "A method of matrix analysis of group structure". Psychometrika 14 (1): 95–116. doi:10.1007/BF02289146. PMID 18152948.. This measure gives an indication of the clustering in the whole network (global), and can be applied to both undirected and directed networks (often called transitivity, see Wasserman and Faust, 1994, page 243Stanley Wasserman, Kathrine Faust, 1994. Social Network Analysis: Methods and Applications. Cambridge: Cambridge University Press.). Formally, it has been defined as: C = \frac{3 \times \mbox{number of triangles}}{\mbox{number of connected triples of vertices}} = \frac{\mbox{number of closed triplets}}{\mbox{number of connected triples of vertices}}.A generalisation to weighted networks was proposed by Opsahl and Panzarasa (2009)Tore Opsahl and Pietro Panzarasa (2009). "Clustering in Weighted Networks". Social Networks 31 (2): 155–163. doi:10.1016/j.socnet.2009.02.002. ., and a redefinition to two-mode networks (both binary and weighted) by Opsahl (2009)Tore Opsahl (2009). "Clustering in Two-mode Networks". Conference and Workshop on Two-Mode Social Analysis (Sept 30-Oct 2, 2009). .. Local clustering coefficient Example local clustering coefficient on an undirected graph. The local clustering coefficient of the light blue node is computed as the proportion of connections among its neighbors which are actually realized compared with the number of all possible connections. In the figure, the light blue node has three neighbours, which can have a maximum of 3 connections among them. In the top part of the figure all three possible connections are realised (thick black segments), giving a local clustering coefficient of 1. In the middle part of the figure only one connection is realized (thick black line) and 2 connections are missing (dotted red lines), giving a local cluster coefficient of 1/3. Finally, none of the possible connections among the neighbours of the light blue node are realised, producing a local clustering coefficient value of 0. The local clustering coefficient of a vertex (graph theory)vertex in a Graph (mathematics)graph quantifies how close its Neighbourhood (graph theory)neighbors are to being a Clique (graph theory)clique (complete graph). Duncan J. Watts and Steven Strogatz introduced the measure in 1998 to determine whether a graph is a small-world network.A graph G=(V,E) formally consists of a set of vertices V and a set of edges E between them. An edge e_{ij} connects vertex i with vertex j. The Neighbourhood (graph theory)neighbourhood N for a vertex v_i is defined as its immediately connected neighbours as follows:N_i = \{v_j : e_{ij} \in E \and e_{ji} \in E\}.The degree (mathematics)degree k_i of a vertex is defined as the number of vertices, |N_i|, in its neighbourhood N_i. The local clustering coefficient C_i for a vertex v_i is then given by the proportion of links between the vertices within its neighbourhood divided by the number of links that could possibly exist between them. For a directed graph, e_{ij} is distinct from e_{ji}, and therefore for each neighbourhood N_i there are k_i(k_i-1) links that could exist among the vertices within the neighbourhood (k_i is the total (in + out) degree of the vertex). Thus, the local clustering coefficient for directed graphs is given asC_i = \frac{|\{e_{jk}\}|}{k_i(k_i-1)} : v_j,v_k \in N_i, e_{jk} \in E.An undirected graph has the property that e_{ij} and e_{ji} are considered identical. Therefore, if a vertex v_i has k_i neighbours, \frac{k_i(k_i-1)}{2} edges could exist among the vertices within the neighbourhood. Thus, the local clustering coefficient for undirected graphs can be defined asC_i = \frac{2|\{e_{jk}\}|}{k_i(k_i-1)} : v_j,v_k \in N_i, e_{jk} \in E.Let \lambda_G(v) be the number of triangles on v \in V(G) for undirected graph G. That is, \lambda_G(v) is the number of subgraphs of G with 3 edges and 3 vertices, one of which is v. Let \tau_G(v) be the number of triples on v \in G. That is, \tau_G(v) is the number of subgraphs (not necessarily induced) with 2 edges and 3 vertices, one of which is v and such that v is incident to both edges. Then we can also define the clustering coefficient as C_i =

\frac{\lambda_G(v)}{\tau_G(v)}.It is simple to show that the two preceding definitions are the same, since \tau_G(v) = C({k_i},2) = \frac{1}{2}k_i(k_i-1).These measures are 1 if every neighbour connected to v_i is also connected to every other vertex within the neighbourhood, and 0 if no vertex that is connected to v_i connects to any other vertex that is connected to v_i. Network average clustering coefficient The clustering coefficient for the whole network is given by Watts and Strogatz as the average of the local clustering coefficients of all the vertices n : \bar{C} = \frac{1}{n}\sum_{i=1}^{n} C_i.A graph is considered Small-world networksmall-world, if its average clustering coefficient \bar{C} is significantly higher than a random graph constructed on the same vertex set, and if the graph has approximately the same distance (graph theory)mean-shortest path length as its corresponding random graph. A generalisation to weighted networks was proposed by Barrat et al. (2004)A. Barrat and M. Barthelemy and R. Pastor-Satorras and A. Vespignani (2004). "The architecture of complex weighted networks". Proceedings of the National Academy of Sciences 101 (11): 3747–3752. doi:10.1073/pnas.0400087101. PMID 15007165.
PMC 374315., and a redefinition to bipartite graphs (also called two-mode networks) by Latapy et al. (2008)M. Latapy and C. Magnien and N. Del Vecchio (2008). "Basic Notions for the Analysis of Large Two-mode Networks". Social Networks 30 (1): 31–48. doi:10.1016/j.socnet.2007.04.006. and Opsahl (2009)Tore Opsahl (2009). "Clustering in Two-mode Networks"
Conference and Workshop on Two-Mode Social Analysis (Sept 30-Oct 2, 2009). .. This formula is not, by default, defined for graphs with isolated vertices; see Kaiser, (2008)Marcus kaiser (2008). "Mean clustering coefficients: the role of isolated nodes and leafs on clustering measures for small-world networks". New Journal of Physics 10 (8): 083042. doi:10.1088/1367-2630/10/8/083042. . and Barmpoutis et al. D.Barmpoutis and R.M. Murray (2010). "Networks with the Smallest Average Distance and the Largest Average Clustering". ArXiv Digital Library. . The networks with the largest possible average clustering coefficient are found to have a modular structure, and at the same time, they have the smallest possible average distance among the different nodes.

Structural cohesion

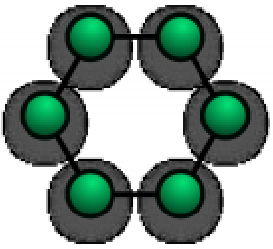
Structural cohesion is the sociological and graph theory conception and measurement of cohesion for maximal social group or graphical boundaries where related elements cannot be disconnected except by removal of a certain minimal number of other nodes. The solution to the boundary problem for structural cohesion is found by the vertex-cut version of Menger's theorem. The boundaries of structural endogamy are a special case of structural cohesion. It is also useful to know that k-cohesive graphs (or k-components) are always a subgraph of a k-core, although a k-core is not always k-cohesive. A k-core is simply a subgraph in which all nodes have at least k neighbors but it need not even be connected.

Software

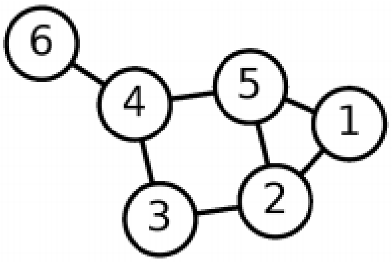
Cohesive.blocking is the R program for computing structural cohesion according to the Moody-White (2003) algorithm. This wiki site provides numerous examples and a tutorial for use with R.

Examples

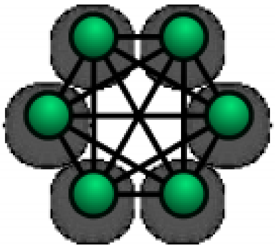
Some illustrative examples are presented in the gallery below:



The 6-node ring in the graph has connectivity-2 or a level 2 of structural cohesion because the removal of two nodes is needed to disconnect it.



The 6-node component (1-connected) has an embedded 2-component, nodes 1-5



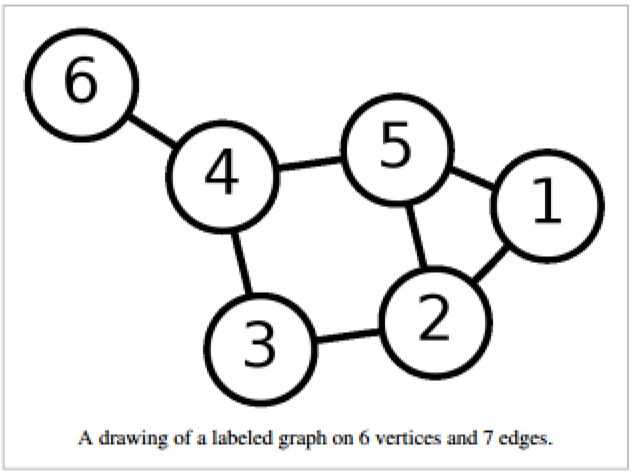
A 6-node clique is a 5-component, structural cohesion 5

Perceived cohesion

Perceived Cohesion Scale (PCS) is a six item scale that is used to measure structural cohesion in groups. In 1990, Bollen and Hoyle used the PCS and applied it to a study of large groups which were used to assess the psychometric qualities of their scale.

Mathematics of Graphs

Graph (mathematics)



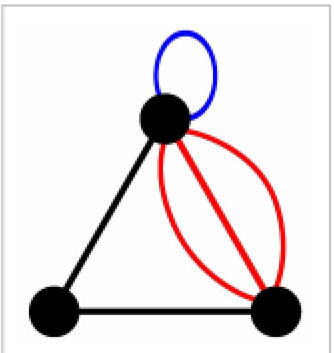
A drawing of a labeled graph on 6 vertices and 7 edges.

In mathematics, a graph is an abstract representation of a set of objects where some pairs of the objects are connected by links. The interconnected objects are represented by mathematical abstractions called vertices, and the links that connect some pairs of vertices are called edges. Typically, a graph is depicted in diagrammatic form as a set of dots for the vertices, joined by lines or curves for the edges. Graphs are one of the objects of study in discrete mathematics. The edges may be directed (asymmetric) or undirected (symmetric). For example, if the vertices represent people at a party, and there is an edge between two people if they shake hands, then this is an undirected graph, because if person A shook hands with person B, then person B also shook hands with person A. On the other hand, if the vertices represent people at a party, and there is an edge from person A to person B when person A knows of person B, then this graph is directed, because knowing of someone is not necessarily a symmetric relation (that is, one person knowing of another person does not necessarily imply the reverse; for example, many fans may know of a celebrity, but the celebrity is unlikely to know of all their fans). This latter type of graph is called a directed graph and the edges are called directed edges or arcs; in contrast, a graph where the edges are not directed is called undirected. Vertices are also called nodes or points, and edges are also called lines. Graphs are the basic subject studied by graph theory. The word "graph" was first used in this sense by James Joseph Sylvester in 1878.

Definitions

Definitions in graph theory vary. The following are some of the more basic ways of defining graphs and related mathematical structures.

Graph


A general example of a graph (actually, a pseudograph) with three vertices and six edges.

In the most common sense of the term,a graph is an ordered pair G = (V, E) comprising a set V of vertices or nodes together with a set E of edges or lines, which are 2-element subsets of V (i.e., an edge is related with two vertices, and the relation is represented as unordered pair of the vertices with respect to the particular edge). To avoid ambiguity, this type of graph may be described precisely as undirected and simple.

Other senses of graph stem from different conceptions of the edge set. In one more generalized notion, E is a set together with a relation of incidence that associates with each edge two vertices. In another generalized notion, E is a multiset of unordered pairs of (not necessarily distinct) vertices. Many authors call this type of object a multigraph or pseudograph.

All of these variants and others are described more fully below.

The vertices belonging to an edge are called the ends, endpoints, or end

vertices of the edge. A vertex may exist in a graph and not belong to an edge.
V and E are usually taken to be finite, and many of the well-known results are not true (or are rather different) for infinite graphs because many of the arguments fail in the infinite case.
The order of a graph is (the number of vertices). A graph's size is , the number of edges. The degree of a vertex is the number of edges that connect to it, where an edge that connects to the vertex at both ends (a loop) is counted twice. For an edge {u, v}, graph theorists usually use the somewhat shorter notation uv.

Adjacency relation

The edges E of an undirected graph G induce a symmetric binary relation ~ on V that is called the adjacency relation of G. Specifically, for each edge {u, v} the vertices u and v are said to be adjacent to one another, which is denoted u ~ v.

Types of graphs

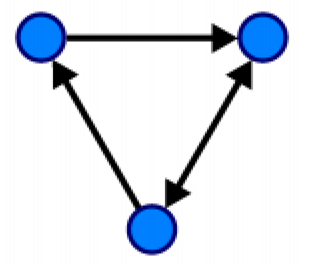
Distinction in terms of the main definition
As stated above, in different contexts it may be useful to define the term graph with different degrees of generality. Whenever it is necessary to draw a strict distinction, the following terms are used. Most commonly, in modern texts in graph theory, unless stated otherwise, graph means "undirected simple finite graph" (see the definitions below).

Undirected graph

A graph in which edges have no orientation, i.e., they are not ordered pairs, but sets {u, v} (or 2-multisets) of vertices.

Directed graph

A directed graph or digraph is an ordered pair D = (V, A) with
• V a set whose elements are called vertices or nodes, and
• A a set of ordered pairs of vertices, called arcs, directed edges, or arrows. An arc a = (x, y) is considered to be directed from x to y; y is called the head and x is called the tail of the arc; y is said to be a direct successor of x, and x is said to be a direct predecessor of y. If a path leads from x to y, then y is said to be a successor of x and reachable from x, and x is said to be a predecessor of y. The arc (y, x) is called the arc (x, y) inverted. A directed graph D is called symmetric if, for every arc in D, the corresponding inverted arc also belongs to D. A symmetric loopless directed graph D = (V, A) is equivalent to a simple undirected graph G = (V, E), where the pairs of inverse arcs in A correspond 1-to-1 with the edges in E; thus the edges in G number |E| = |A|/2, or half the number of arcs in D.


A directed graph.

A variation on this definition is the oriented graph, in which not more than one of (x, y) and (y, x) may be arcs.

Mixed graph

A mixed graph G is a graph in which some edges may be directed and some may be undirected. It is written as an ordered triple G = (V, E, A) with V, E, and A defined as above. Directed and undirected graphs are special cases.

Multigraph

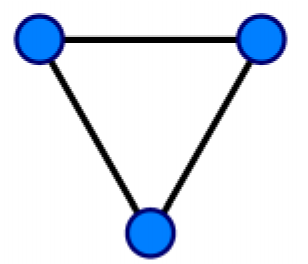
A loop is an edge (directed or undirected) which starts and ends on the same vertex; these may be permitted or not permitted according to the application. In this context, an edge with two different ends is called a link.

The term "multigraph" is generally understood to mean that multiple edges (and sometimes loops) are allowed. Where graphs are defined so as to allow loops and multiple edges, a multigraph is often defined to mean a graph without loops, however, where graphs are defined so as to disallow loops and multiple edges, the term is often defined to mean a "graph" which can have both multiple edges and loops, although many use the term "pseudograph" for this meaning.

Simple graph

A simple graph with three vertices and three edges. Each vertex has degree two, so this is also a regular graph.

As opposed to a multigraph, a simple graph is an undirected graph that has no loops and no more than one edge between any two different vertices. In a simple graph the edges of the graph form a set (rather than a multiset) and each edge is a pair of distinct vertices. In a simple graph with n vertices every vertex has a degree that is less than n (the converse, however, is not true - there exist non-simple graphs with n vertices in which every vertex has a degree smaller than n).



A simple graph with three vertices and three edges. Each vertex has degree two, so this is also a regular graph.

Weighted graph

A graph is a weighted graph if a number (weight) is assigned to each edge. Such weights might represent, for example, costs, lengths or capacities, etc. depending on the problem.

The weight of the graph is the sum of the weights given to all edges.

Half-edges, loose edges

In exceptional situations it is even necessary to have edges with only one end, called half-edges, or no ends (loose edges); see for example signed graphs and biased graphs.

Important graph classes

Regular graph

A regular graph is a graph where each vertex has the same number of neighbors, i.e., every vertex has the same degree or valency. A regular graph with vertices of degree k is called a k-regular graph or regular graph of degree k.

Complete graph

Complete graphs have the feature that each pair of vertices has an edge connecting them.

Finite and infinite graphs

A finite graph is a graph G = (V, E) such that V and E are finite sets. An infinite graph is one with an infinite set of vertices or edges or both. Most commonly in graph theory it is implied that the graphs discussed are finite. If the graphs are infinite, that is usually specifically stated.

Graph classes in terms of connectivity

In an undirected graph G, two vertices u and v are called connected if G contains a path from u to v. Otherwise, they are called disconnected. A graph is called connected if every pair of distinct vertices in the graph is connected; otherwise, it is called disconnected. A graph is called k-vertex-connected or k-edge-connected if no set of k-1 vertices (respectively, edges) exists that disconnects the graph. A k-vertex-connected graph is often called simply k-connected. A directed graph is called weakly connected if replacing all of its directed edges with undirected edges produces a connected (undirected) graph. It is strongly connected or strong if it contains a directed path from u to v and a directed path from v to u for every pair of vertices u, v.

Properties of graphs

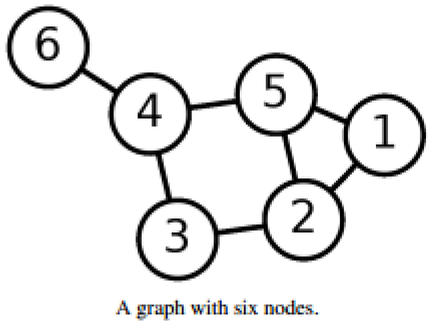
Two edges of a graph are called adjacent (sometimes coincident) if they share a common vertex. Two arrows of a directed graph are called consecutive if the head of the first one is at the nock (notch end) of the second one. Similarly, two vertices are called adjacent if they share a common edge (consecutive if they are at the notch and at the head of an arrow), in which case the common edge is said to join the two vertices. An edge and a vertex on that edge are called incident.

The graph with only one vertex and no edges is called the trivial graph. A graph with only vertices and no edges is known as an edgeless graph. The graph with no vertices and no edges is sometimes called the null graph or empty graph, but the terminology is not consistent and not all mathematicians allow this object. In a weighted graph or digraph, each edge is associated with some value, variously called its cost, weight, length or other term depending on the application; such graphs arise in many contexts, for example in optimal routing problems such as the traveling salesman problem.

Normally, the vertices of a graph, by their nature as elements of a set, are distinguishable. This kind of graph may be called vertex-labeled. However, for many questions it is better to treat vertices as indistinguishable; then the graph may be called unlabeled. (Of course, the vertices may be still distinguishable by the properties of the graph itself, Graph (mathematics) 66 e.g., by the numbers of incident edges). The same remarks apply to edges, so that graphs which have labeled edges are called edge-labeled graphs. Graphs with labels attached to edges or vertices are more generally designated as labeled. Consequently, graphs in which vertices are indistinguishable and edges are indistinguishable are called unlabeled. (Note that in the literature the term labeled may apply to other kinds of labeling, besides that which serves only to distinguish different vertices or edges.)

The diagram at right is a graphic representation of the following graph:
V = {1, 2, 3, 4, 5, 6}
E = .

- In category theory a small category can be represented by a directed multigraph in which the objects of the category represented as vertices and the morphisms as directed edges. Then, the functors between categories induce some, but not necessarily all, of the digraph morphisms of the graph.
- In computer science, directed graphs are used to represent knowledge (e.g., Conceptual graph), finite statemachines, and many other discrete structures.



A graph with six nodes.

- A binary relation R on a set X defines a directed graph. An element x of X is a direct predecessor of an element y of X iff xRy

Important graphs

Basic examples are:

- In a complete graph, each pair of vertices is joined by an edge; that is, the graph contains all possible edges.
- In a bipartite graph, the vertex set can be partitioned into two sets, W and X, so that no two vertices in W are adjacent and no two vertices in X are adjacent. Alternatively, it is a graph with a chromatic number of 2.
- In a complete bipartite graph, the vertex set is the union of two disjoint sets, W and X, so that every vertex in W is adjacent to every vertex in X but there are no edges within W or X.
- In a linear graph or path graph of length n, the vertices can be listed in order, v0, v1 , ..., vn, so that the edges are v i−1 vi for each i = 1, 2, ..., n. If a linear graph occurs as a subgraph of another graph, it is a path in that graph.
- In a cycle graph of length n ≥ 3, vertices can be named v1, ..., vn so that the edges are vi−1 vi for each i = 2,...,n in addition to vnv1. Cycle graphs can be characterized as connected 2-regular graphs. If a cycle graph occurs as a subgraph of another graph, it is a cycle or circuit in that graph.
- A planar graph is a graph whose vertices and edges can be drawn in a plane such that no two of the edges
intersect (i.e., embedded in a plane).
- A tree is a connected graph with no cycles.
- A forest is a graph with no cycles (i.e. the disjoint union of one or more trees).

More advanced kinds of graphs are:

- The Petersen graph and its generalizations
- Perfect graphs
- Cographs
- Other graphs with large automorphism groups: vertex-transitive, arc-transitive, and distance-transitive graphs.
- Strongly regular graphs and their generalization distance-regular graphs.

Operations on graphs

There are several operations that produce new graphs from old ones, which might be classified into the following categories:

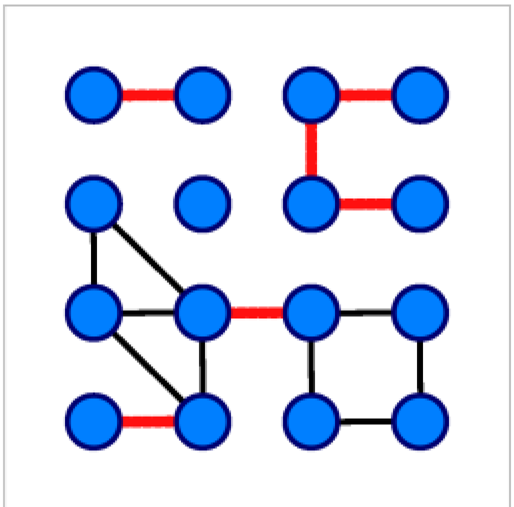
- Elementary operations, sometimes called "editing operations" on graphs, which create a new graph from the original one by a simple, local change, such as addition or deletion of a vertex or an edge, merging and splitting of vertices, etc.
- Graph rewrite operations replacing the occurrence of some pattern graph within the host graph by an instance of the corresponding replacement graph.
- Unary operations, which create a significantly new graph from the old one. Examples:
- Line graph
- Dual graph
- Complement graph
- Binary operations, which create new graph from two initial graphs. Examples:

- Disjoint union of graphs
- Cartesian product of graphs
- Tensor product of graphs
- Strong product of graphs
- Lexicographic product of graphs

Generalizations
In a hypergraph, an edge can join more than two vertices.
An undirected graph can be seen as a simplicial complex consisting of 1-simplices (the edges) and 0-simplices (the vertices). As such, complexes are generalizations of graphs since they allow for higher-dimensional simplices. Every graph gives rise to a matroid. In model theory, a graph is just a structure. But in that case, there is no limitation on the number of edges: it can be any cardinal number, see continuous graph. In computational biology, power graph analysis introduces power graphs as an alternative representation of undirected graphs.

Bridge (graph theory)

In graph theory, a bridge (also known as a cut-edge or cut arc or an isthmus) is an edge whose deletion increases the number of connected components. Equivalently, an edge is a bridge if and only if it is not contained in any cycle.
A graph is said to be bridgeless if it contains no bridges. It is easy to see that this is equivalent to 2-edge-connectivity of each nontrivial component.



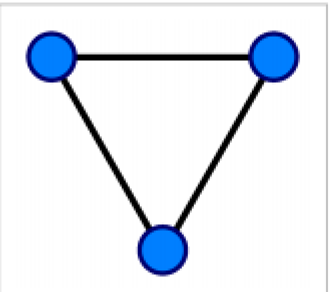
A graph with 6 bridges (highlighted in red)

Cycle double cover conjecture

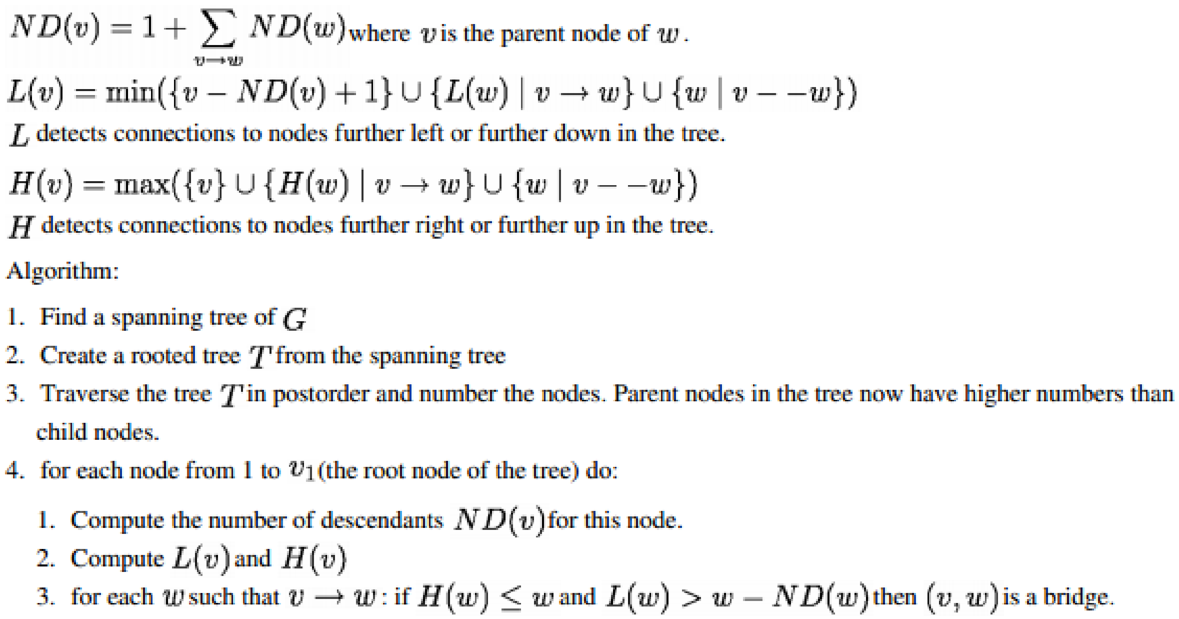
An important open problem involving bridges is the cycle double cover conjecture, due to Seymour and Szekeres (1978 and 1979, independently), which states that every bridgeless graph admits a set of cycles which contains each edge exactly twice.

Bridge-Finding Algorithm

An $O(|V|+|E|)$ algorithm for finding bridges in a connected graph was found by Tarjan in 1974.
Definitions: A non-tree edge between v and w is denoted by $v--w$ An in-tree edge with as the parent v is denoted by $v \to w$.



An undirected connected graph with no cut edges



$ND(v) = 1 + \sum_{v \to w} ND(w)$ where v is the parent node of w.

$L(v) = \min(\{v - ND(v) + 1\} \cup \{L(w) \mid v \to w\} \cup \{w \mid v - -w\})$

L detects connections to nodes further left or further down in the tree.

$H(v) = \max(\{v\} \cup \{H(w) \mid v \to w\} \cup \{w \mid v - -w\})$

H detects connections to nodes further right or further up in the tree.

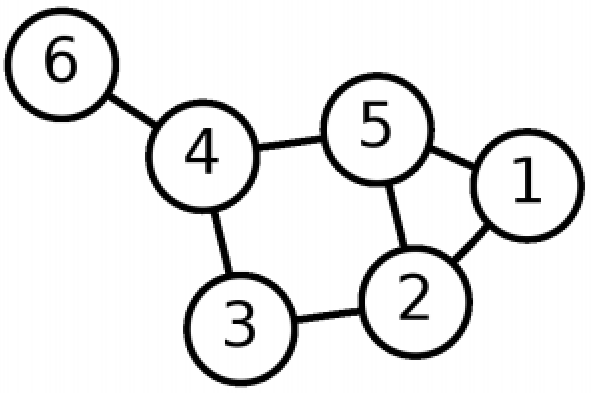
Algorithm:

1. Find a spanning tree of G
2. Create a rooted tree T from the spanning tree
3. Traverse the tree T in postorder and number the nodes. Parent nodes in the tree now have higher numbers than child nodes.
4. for each node from 1 to v_1 (the root node of the tree) do:
    1. Compute the number of descendants $ND(v)$ for this node.
    2. Compute $L(v)$ and $H(v)$
    3. for each w such that $v \to w$: if $H(w) \leq w$ and $L(w) > w - ND(w)$ then (v, w) is a bridge.

Cut arc in trees

An edge or arc e = uv of a tree G is a cut arc of G if and only if the degree of the vertices u and v are greater than 1. Cut arcs are also defined for directed graphs.

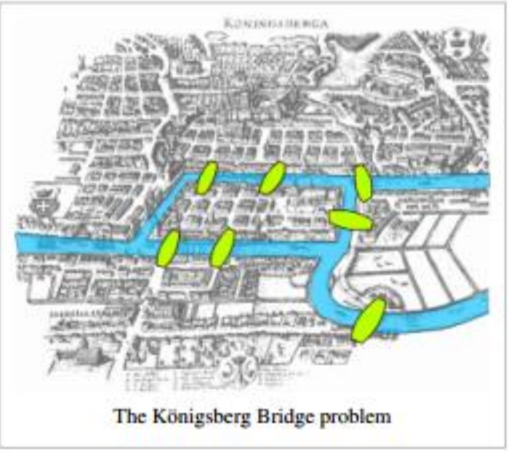
Graph theory



A drawing of a graph

In mathematics and computer science, graph theory is the study of graphs: mathematical structures used to model pairwise relations between objects from a certain collection. A "graph" in this context refers to a collection of vertices or 'nodes' and a collection of edges that connect pairs of vertices. A graph may be undirected, meaning that there is no distinction between the two vertices associated with each edge, or its edges may be directed from one vertex to another; see graph (mathematics) for more detailed definitions and for other variations in the types of graphs that are commonly considered. The graphs studied in graph theory should not be confused with "graphs of functions" and other kinds of graphs. Graphs are one of the prime objects of study in Discrete Mathematics. Refer to Glossary of graph theory for basic definitions in graph theory.

History



The Königsberg Bridge problem

The paper written by Leonhard Euler on the Seven Bridges of Königsberg and published in 1736 is regarded as the first paper in the history of graph theory. This paper, as well as the one written by Vandermonde on the knight problem, carried on with the analysis situs initiated by Leibniz. Euler's formula relating the number of edges, vertices, and faces of a convex polyhedron was studied and generalized by Cauchy and L'Huillier, and is at the origin of topology.

More than one century after Euler's paper on the bridges of Königsberg and while Listing introduced topology, Cayley was led by the study of particular analytical forms arising from differential calculus to study a particular class of graphs, the trees. This study had many implications in theoretical chemistry. The involved techniques mainly concerned the enumeration of graphs having particular properties. Enumerative graph theory then rose from the results of Cayley and the fundamental results published by Pólya between 1935 and 1937 and the generalization of these by De Bruijn in 1959. Cayley linked his results on trees with the contemporary studies of chemical composition. The fusion of the ideas coming from mathematics with those coming from chemistry is at the origin of a part of the standard terminology of graph theory. In particular, the term "graph" was introduced by Sylvester in a paper published in 1878 in Nature, where he draws an analogy between "quantic invariants" and "co-variants" of algebra and molecular diagrams:

"[...] Every invariant and co-variant thus becomes expressible by a graph precisely identical with a Kekuléan diagram or chemicograph. [...] I give a rule for the geometrical multiplication of graphs, i.e. for constructing a graph to the product of in- or co-variants whose separate graphs are given. [...]" (italics as in the original). One of the most famous and productive problems of graph theory is the four color problem: "Is it true that any map drawn in the plane may have its regions colored with four colors, in such a way that any two regions having a common border have different colors?" This problem was first posed by Francis Guthrie in 1852 and its first written record is in a letter of De Morgan addressed to Hamilton the same year. Many incorrect proofs have been proposed, including those by Cayley, Kempe, and others. The study and the generalization of this problem by Tait, Heawood, Ramsey and Hadwiger led to the study of the colorings of the graphs embedded on surfaces with arbitrary genus. Tait's reformulation generated a new class of problems, the factorization problems, particularly studied by Petersen and Kőnig. The works of Ramsey on colorations and more specially the results obtained by Turán in 1941 was at the origin of another branch of graph theory, extremal graph theory. The four color problem remained unsolved for more than a century. In 1969 Heinrich Heesch published a method for solving the problem using computers. A computer-aided proof produced in 1976 by Kenneth Appel and Wolfgang Haken makes fundamental use of the notion of "discharging" developed by Heesch.

The proof involved checking the properties of 1,936 configurations by computer, and was not fully accepted at the time due to its complexity. A simpler proof considering only 633 configurations was given twenty years later by Robertson, Seymour, Sanders and Thomas. The autonomous development of topology from 1860 and 1930 fertilized graph theory back through the works of Jordan, Kuratowski and Whitney. Another important factor of common development of graph theory and topology came from the use of the techniques of modern algebra. The first example of such a use comes from the work of the physicist Gustav Kirchhoff, who published in 1845 his Kirchhoff's circuit laws for calculating the voltage and current in electric circuits. The introduction of probabilistic methods in graph theory, especially in the study of Erdős and Rényi of the asymptotic probability of graph connectivity, gave rise to yet another branch, known as random graph theory, which has been a fruitful source of graph-theoretic results.

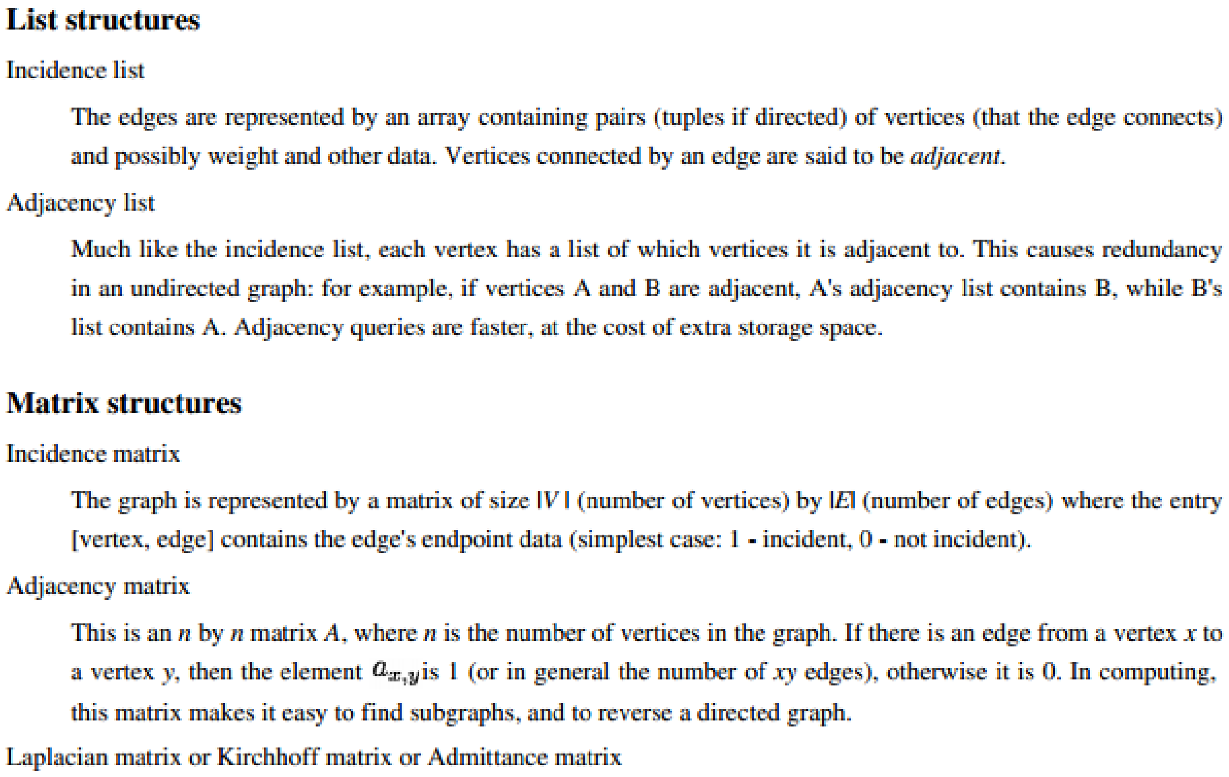
Drawing graphs

Graphs are represented graphically by drawing a dot for every vertex, and drawing an arc between two vertices if they are connected by an edge. If the graph is directed, the direction is indicated by drawing an arrow. A graph drawing should not be confused with the graph itself (the abstract, non-visual structure) as there are several ways to structure the graph drawing. All that matters is which vertices are connected to which others by how many edges and not the exact layout. In practice it is often difficult to decide if two drawings represent the same graph. Depending on the problem domain some layouts may be better suited and easier to understand than others.

Graph-theoretic data structures

There are different ways to store graphs in a computer system. The data structure used depends on both the graph structure and the algorithm used for manipulating the graph. Theoretically one can distinguish between list and matrix structures but in concrete applications the best structure is often a combination of both. List structures are

often preferred for sparse graphs as they have smaller memory requirements. Matrix structures on the other hand provide faster access for some applications but can consume huge amounts of memory.



List structures

Incidence list

> The edges are represented by an array containing pairs (tuples if directed) of vertices (that the edge connects) and possibly weight and other data. Vertices connected by an edge are said to be *adjacent*.

Adjacency list

> Much like the incidence list, each vertex has a list of which vertices it is adjacent to. This causes redundancy in an undirected graph: for example, if vertices A and B are adjacent, A's adjacency list contains B, while B's list contains A. Adjacency queries are faster, at the cost of extra storage space.

Matrix structures

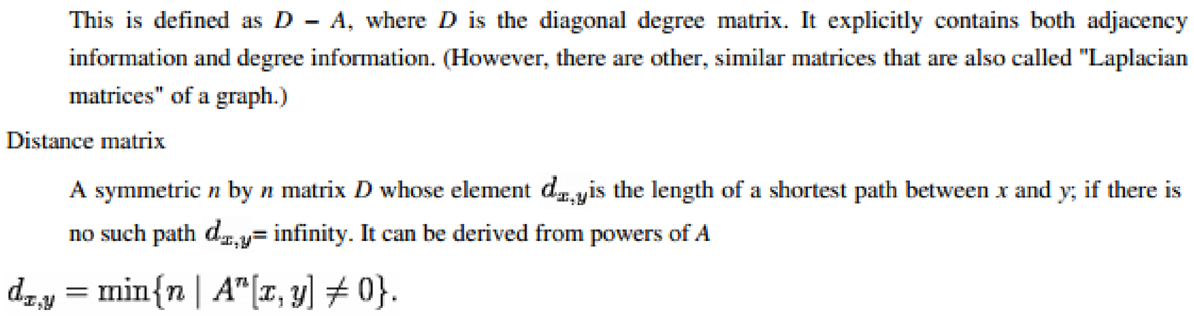
Incidence matrix

> The graph is represented by a matrix of size $|V|$ (number of vertices) by $|E|$ (number of edges) where the entry [vertex, edge] contains the edge's endpoint data (simplest case: 1 - incident, 0 - not incident).

Adjacency matrix

> This is an n by n matrix A, where n is the number of vertices in the graph. If there is an edge from a vertex x to a vertex y, then the element $a_{x,y}$is 1 (or in general the number of xy edges), otherwise it is 0. In computing, this matrix makes it easy to find subgraphs, and to reverse a directed graph.

Laplacian matrix or Kirchhoff matrix or Admittance matrix



This is defined as $D - A$, where D is the diagonal degree matrix. It explicitly contains both adjacency information and degree information. (However, there are other, similar matrices that are also called "Laplacian matrices" of a graph.)

Distance matrix

A symmetric n by n matrix D whose element $d_{x,y}$is the length of a shortest path between x and y; if there is no such path $d_{x,y}$= infinity. It can be derived from powers of A

$$d_{x,y} = \min\{n \mid A^n[x,y] \neq 0\}.$$

Problems in graph theory

Enumeration

There is a large literature on graphical enumeration: the problem of counting graphs meeting specified conditions. Some of this work is found in Harary and Palmer (1973).

Subgraphs, induced subgraphs, and minors

A common problem, called the subgraph isomorphism problem, is finding a fixed graph as a subgraph in a given graph. One reason to be interested in such a question is that many graph properties are hereditary for subgraphs, which means that a graph has the property if and only if all subgraphs have it too. Unfortunately, finding maximal subgraphs of a certain kind is often an NP-complete problem.

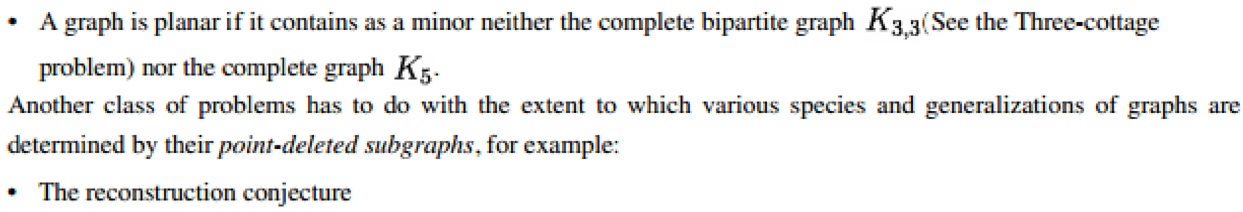
- Finding the largest complete graph is called the clique problem (NP-complete).

A similar problem is finding induced subgraphs in a given graph. Again, some important graph properties are hereditary with respect to induced subgraphs, which means that a graph has a property if and only if all induced subgraphs also have it. Finding maximal induced subgraphs of a certain kind is also often NP-complete. For example,

- Finding the largest edgeless induced subgraph, or independent set, called the independent set problem (NP-complete).

Still another such problem, the minor containment problem, is to find a fixed graph as a minor of a given graph. A minor or subcontraction of a graph is any graph obtained by taking a subgraph and contracting some (or no) edges.

Many graph properties are hereditary for minors, which means that a graph has a property if and only if all minors have it too. A famous example:



- A graph is planar if it contains as a minor neither the complete bipartite graph $K_{3,3}$(See the Three-cottage problem) nor the complete graph K_5.

Another class of problems has to do with the extent to which various species and generalizations of graphs are determined by their *point-deleted subgraphs*, for example:

- The reconstruction conjecture

Graph coloring

Many problems have to do with various ways of coloring graphs, for example:
- The four-color theorem
- The strong perfect graph theorem
- The Erdős–Faber–Lovász conjecture (unsolved)
- The total coloring conjecture (unsolved)
- The list coloring conjecture (unsolved)
- The Hadwiger conjecture (graph theory) (unsolved)

Route problems

- Hamiltonian path and cycle problems
- Minimum spanning tree
- Route inspection problem (also called the "Chinese Postman Problem")
- Seven Bridges of Königsberg
- Shortest path problem
- Steiner tree
- Three-cottage problem
- Traveling salesman problem (NP-complete)

Network flow

There are numerous problems arising especially from applications that have to do with various notions of flows in networks, for example:
- Max flow min cut theorem

Visibility graph problems

- Museum guard problem

Covering problems

Covering problems are specific instances of subgraph-finding problems, and they tend to be closely related to the clique problem or the independent set problem.
- Set cover problem
- Vertex cover problem

Graph classes

Many problems involve characterizing the members of various classes of graphs. Overlapping significantly with other types in this list, this type of problem includes, for instance:
- Enumerating the members of a class
- Characterizing a class in terms of forbidden substructures
- Ascertaining relationships among classes (e.g., does one property of graphs imply another)
- Finding efficient algorithms to decide membership in a class
- Finding representations for members of a class

Applications

Graphs are among the most ubiquitous models of both natural and human-made structures. They can be used to model many types of relations and process dynamics in physical, biological and social systems. Many problems of practical interest can be represented by graphs.

In computer science, graphs are used to represent networks of communication, data organization, computational devices, the flow of computation, etc. One practical example: The link structure of a website could be represented by a directed graph. The vertices are the web pages available at the website and a directed edge from page A to page B exists if and only if A contains a link to B. A similar approach can be taken to problems in travel, biology, computer chip design, and many other fields. The development of algorithms to handle graphs is therefore of major interest in computer science. There, the transformation of graphs is often formalized and represented by graph rewrite systems.
They are either directly used or properties of the rewrite systems (e.g. confluence) are studied. Complementary to graph transformation systems focussing on rule-based in-memory manipulation of graphs are graph databases geared towards transaction-safe, persistent storing and querying of graph-structured data.
Graph-theoretic methods, in various forms, have proven particularly useful in linguistics, since natural language often lends itself well to discrete structure. Traditionally, syntax and compositional semantics follow tree-based structures, whose expressive power lies in the Principle of Compositionality, modeled in a hierarchical graph. Within lexical semantics, especially as applied to computers, modeling word meaning is easier when a given word is understood in terms of related words; semantic networks are therefore important in computational linguistics. Still other methods in phonology (e.g. Optimality Theory, which uses lattice graphs) and morphology (e.g. finite-state morphology, using finite-state transducers) are common in the analysis of language as a graph. Indeed, the usefulness of this area of mathematics to linguistics has borne organizations such as TextGraphs [10], as well as various 'Net' projects, such as WordNet, VerbNet, and others.
Graph theory is also used to study molecules in chemistry and physics. In condensed matter physics, the three dimensional structure of complicated simulated atomic structures can be studied quantitatively by gathering statistics on graph-theoretic properties related to the topology of the atoms. For example, Franzblau's shortest-path (SP) rings. In chemistry a graph makes a natural model for a molecule, where vertices represent atoms and edges bonds. This approach is especially used in computer processing of molecular structures, ranging from chemical editors to database searching. In statistical physics, graphs can represent local connections between interacting parts of a system, as well as the dynamics of a physical process on such systems.
Graph theory is also widely used in sociology as a way, for example, to measure actors' prestige or to explore diffusion mechanisms, notably through the use of social network analysis software. Likewise, graph theory is useful in biology and conservation efforts where a vertex can represent regions where certain species exist (or habitats) and the edges represent migration paths, or movement between the regions. This information is important when looking at breeding patterns or tracking the spread of disease, parasites or how changes to the movement can affect other species. In mathematics, graphs are useful in geometry and certain parts of topology, e.g. Knot Theory. Algebraic graph theory has close links with group theory. A graph structure can be extended by assigning a weight to each edge of the graph. Graphs with weights, or weighted graphs, are used to represent structures in which pairwise connections have some numerical values. For example if a graph represents a road network, the weights could represent the length of each road. A digraph with weighted edges in the context of graph theory is called a network. Network analysis have many practical applications, for example, to model and analyze traffic networks. Applications of network analysis split broadly into three categories:
1. First, analysis to determine structural properties of a network, such as the distribution of vertex degrees and the diameter of the graph. A vast number of graph measures exist, and the production of useful ones for various domains remains an active area of research.

2. Second, analysis to find a measurable quantity within the network, for example, for a transportation network, the level of vehicular flow within any portion of it.
3. Third, analysis of dynamical properties of networks.

Network theory

For the sociological theory, see Social network

Network theory is an area of computer science and network science and part of graph theory. It has application in many disciplines including particle physics, computer science, biology, economics, operations research, and sociology. Network theory concerns itself with the study of graphs as a representation of either symmetric relations or, more generally, of asymmetric relations between discrete objects. Applications of network theory include logistical networks, the World Wide Web, gene regulatory networks, metabolic networks, social networks, epistemological networks, etc. See list of network theory topics for more examples.

Network optimization

Network problems that involve finding an optimal way of doing something are studied under the name of combinatorial optimization. Examples include network flow, shortest path problem, transport problem, transshipment problem, location problem, matching problem, assignment problem, packing problem, routing problem, Critical Path Analysis and PERT (Program Evaluation & Review Technique).

Network analysis

Social network analysis

Social network analysis maps relationships between individuals in social networks. Such individuals are often persons, but may be groups (including cliques and cohesive blocks), organizations, nation states, web sites, or citations between scholarly publications (scientometrics). Network analysis, and its close cousin traffic analysis, has significant use in intelligence. By monitoring the communication patterns between the network nodes, its structure can be established. This can be used for uncovering insurgent networks of both hierarchical and leaderless nature.

Biological network analysis

With the recent explosion of publicly available high throughput biological data, the analysis of molecular networks has gained significant interest. The type of analysis in this content are closely related to social network analysis, but often focusing on local patterns in the network. For example network motifs are small subgraphs that are over-represented in the network. Activity motifs are similar over-represented patterns in the attributes of nodes and edges in the network that are over represented given the network structure.

Link analysis

Link analysis is a subset of network analysis, exploring associations between objects. An example may be examining the addresses of suspects and victims, the telephone numbers they have dialed and financial transactions that they have partaken in during a given timeframe, and the familial relationships between these subjects as a part of police investigation. Link analysis here provides the crucial relationships and associations between very many objects of different types that are not apparent from isolated pieces of information.

Computer-assisted or fully automatic

computer-based link analysis is increasingly employed by banks and insurance agencies in fraud detection, by telecommunication operators in telecommunication network analysis, by medical sector in epidemiology and pharmacology, in law enforcement investigations, by search engines for relevance rating (and conversely by the spammers for spamdexing and by business owners for search engine optimization), and everywhere else where relationships between many objects have to be analyzed.

Web link analysis
Several Web search ranking algorithms use link-based centrality metrics, including (in order of appearance) Marchiori's Hyper Search, Google's PageRank, Kleinberg's HITS algorithm, and the TrustRank algorithm. Link analysis is also conducted in information science and communication science in order to understand and extract information from the structure of collections of web pages. For example the analysis might be of the interlinking between politicians' web sites or blogs.

Centrality measures
Information about the relative importance of nodes and edges in a graph can be obtained through centrality measures, widely used in disciplines like sociology. For example, eigenvector centrality uses the eigenvectors of the adjacency matrix to determine nodes that tend to be frequently visited.

Spread of content in networks
Content in a complex network can spread via two major methods: conserved spread and non-conserved spread. In conserved spread, the total amount of content that enters a complex network remains constant as it passes through.The model of conserved spread can best be represented by a pitcher containing a fixed amount of water being poured into a series of funnels connected by tubes . Here, the pitcher represents the original source and the water is the content being spread. The funnels and connecting tubing represent the nodes and the connections between nodes, respectively. As the water passes from one funnel into another, the water disappears instantly from the funnel that was previously exposed to the water. In non-conserved spread, the amount of content changes as it enters and passes through a complex network. The model of non-conserved spread can best be represented by a continuously running faucet running through a series of funnels connected by tubes. Here, the amount of water from the original source is infinite. Also, any funnels that have been exposed to the water continue to experience the water even as it passes into successive funnels. The non-conserved model is the most suitable for explaining the transmission of most infectious diseases.

Implementations

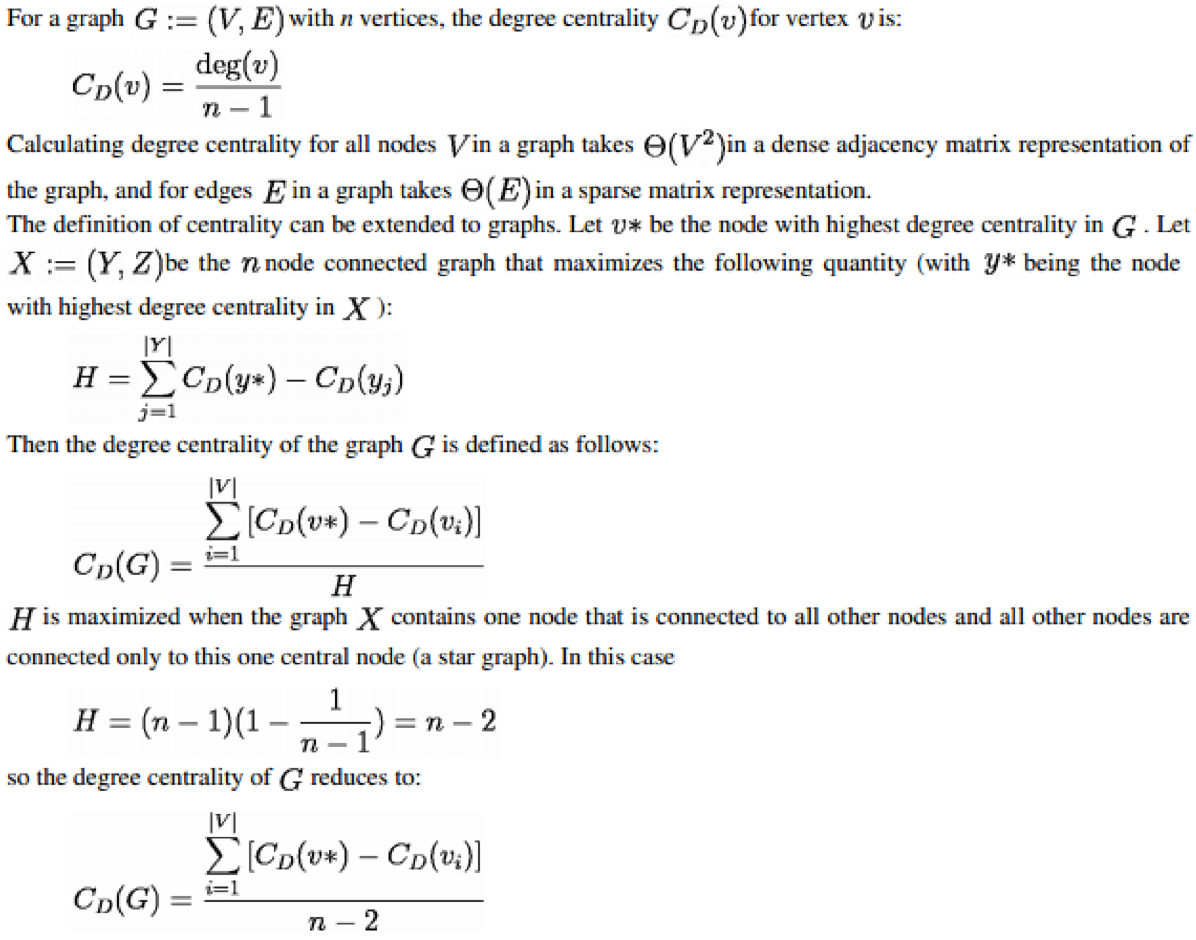
- Orange, a free data mining software suite, module orngNetwork
- Pajek ,program for (large) network analysis and visualization
- Tulip, a free data mining and visualization software dedicated to the analysis and visualization of relational data.

Closeness (graph theory)

Within graph theory and network analysis, there are various measures of the centrality of a vertex within a graph that determine the relative importance of a vertex within the graph (for example, how important a person is within a social network, or, in the theory of space syntax, how important a room is within a building or how well-used a road is within an urban network).
There are four measures of centrality that are widely used in network analysis: degree centrality, betweenness, closeness, and eigenvector centrality. For a review as well as generalizations to weighted networks, see Opsahl et al. (2010)

Degree centrality

The first, and simplest, is degree centrality. Degree centrality is defined as the number of links incident upon a node (i.e., the number of ties that a node has). Degree is often interpreted in terms of the immediate risk of node for catching whatever is flowing through the network (such as a virus, or some information). If the network is directed (meaning that ties have direction), then we usually define two separate measures of degree centrality, namely indegree and outdegree. Indegree is a count of the number of ties directed to the node, and outdegree is the number of ties that the node directs to others. For positive relations such as friendship or advice, we normally interpret indegree as a form of popularity, and outdegree as gregariousness.



For a graph $G := (V, E)$ with n vertices, the degree centrality $C_D(v)$ for vertex v is:

$$C_D(v) = \frac{\deg(v)}{n-1}$$

Calculating degree centrality for all nodes V in a graph takes $\Theta(V^2)$ in a dense adjacency matrix representation of the graph, and for edges E in a graph takes $\Theta(E)$ in a sparse matrix representation.

The definition of centrality can be extended to graphs. Let $v*$ be the node with highest degree centrality in G. Let $X := (Y, Z)$ be the n node connected graph that maximizes the following quantity (with $y*$ being the node with highest degree centrality in X):

$$H = \sum_{j=1}^{|Y|} C_D(y*) - C_D(y_j)$$

Then the degree centrality of the graph G is defined as follows:

$$C_D(G) = \frac{\sum_{i=1}^{|V|} [C_D(v*) - C_D(v_i)]}{H}$$

H is maximized when the graph X contains one node that is connected to all other nodes and all other nodes are connected only to this one central node (a star graph). In this case

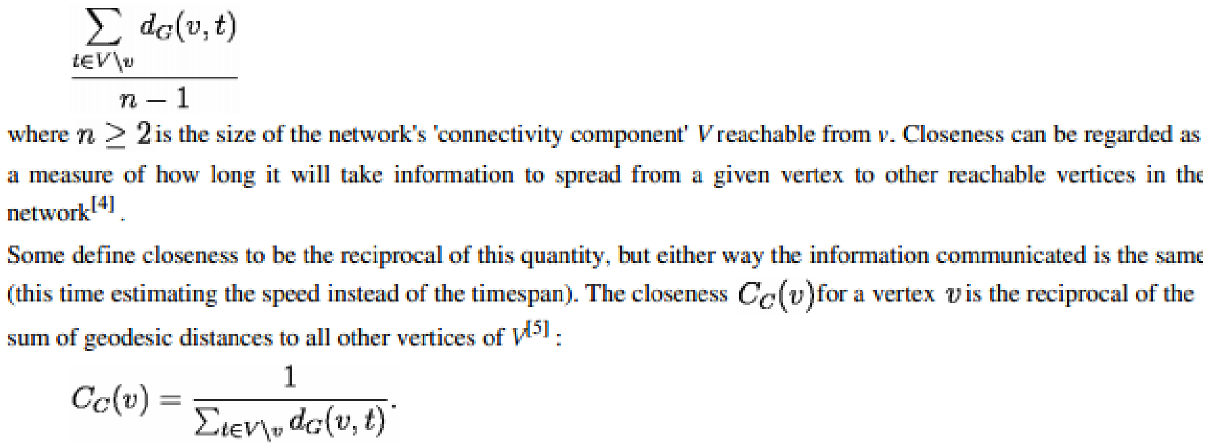
$$H = (n-1)(1 - \frac{1}{n-1}) = n - 2$$

so the degree centrality of G reduces to:

$$C_D(G) = \frac{\sum_{i=1}^{|V|} [C_D(v*) - C_D(v_i)]}{n-2}$$

Closeness centrality

In topology and related areas in mathematics, closeness is one of the basic concepts in a topological space. Intuitively we say two sets are close if they are arbitrarily near to each other. The concept can be defined naturally in a metric space where a notion of distance between elements of the space is defined, but it can be generalized to topological spaces where we have no concrete way to measure distances. In graph theory closeness is a centrality measure of a vertex within a graph. Vertices that are 'shallow' to other vertices (that is, those that tend to have short geodesic distances to other vertices with in the graph) have higher closeness. Closeness is preferred in network analysis to mean shortest-path length, as it gives higher values to more central vertices, and so is usually positively associated with other measures such as degree. In the network theory, closeness is a sophisticated measure of centrality. It is defined as the mean geodesic distance (i.e., the shortest path) between a vertex v and all other vertices reachable from it:



$$\frac{\sum_{t \in V \setminus v} d_G(v,t)}{n-1}$$

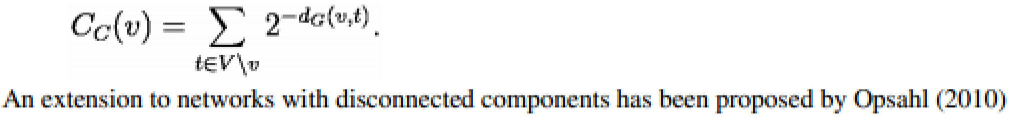
where $n \geq 2$ is the size of the network's 'connectivity component' V reachable from v. Closeness can be regarded as a measure of how long it will take information to spread from a given vertex to other reachable vertices in the network[4] .

Some define closeness to be the reciprocal of this quantity, but either way the information communicated is the same (this time estimating the speed instead of the timespan). The closeness $C_C(v)$ for a vertex v is the reciprocal of the sum of geodesic distances to all other vertices of V[5] :

$$C_C(v) = \frac{1}{\sum_{t \in V \setminus v} d_G(v,t)}.$$

Different methods and algorithms can be introduced to measure closeness, like the random-walk centrality introduced by Noh and Rieger (2003) that is a measure of the speed with which randomly walking messages reach a vertex from elsewhere in the network—a sort of random-walk version of closeness centrality.

The information centrality of Stephenson and Zelen (1989) is another closeness measure, which bears some similarity to that of Noh and Rieger. In essence it measures the harmonic mean length of paths ending at a vertex i, which is smaller if i has many short paths connecting it to other vertices. Dangalchev (2006), in order to measure the network vulnerability, modifies the definition for closeness so it can be used for disconnected graphs and the total closeness is easier to calculate:



$$C_C(v) = \sum_{t \in V \setminus v} 2^{-d_G(v,t)}.$$

An extension to networks with disconnected components has been proposed by Opsahl (2010)

Eigenvector centrality

Eigenvector centrality is a measure of the importance of a node in a network. It assigns relative scores to all nodes in the network based on the principle that connections to high-scoring nodes contribute more to the score of the node in question than equal connections to low-scoring nodes. Google's PageRank is a variant of the Eigenvector centrality measure.

Using the adjacency matrix to find eigenvector centrality

Let x_i denote the score of the *i*th node. Let $A_{i,j}$ be the adjacency matrix of the network. Hence $A_{i,j} = 1$if the *i*th node is adjacent to the *j*th node, and $A_{i,j} = 0$otherwise. More generally, the entries in *A* can be real numbers representing connection strengths, as in a stochastic matrix.

For the i^{th}node, let the centrality score be proportional to the sum of the scores of all nodes which are connected to it. Hence

$$x_i = \frac{1}{\lambda} \sum_{j \in M(i)} x_j = \frac{1}{\lambda} \sum_{j=1}^{N} A_{i,j} x_j$$

where $M(i)$is the set of nodes that are connected to the i^{th}node, *N* is the total number of nodes and λ is a constant. In vector notation this can be rewritten as

$$\mathbf{x} = \frac{1}{\lambda}\mathbf{A}\mathbf{x}$$, or as the eigenvector equation $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$

In general, there will be many different eigenvalues λ for which an eigenvector solution exists. However, the additional requirement that all the entries in the eigenvector be positive implies (by the Perron–Frobenius theorem) that only the greatest eigenvalue results in the desired centrality measure.[10] The i^{th}component of the related eigenvector then gives the centrality score of the i^{th}node in the network. Power iteration is one of many eigenvalue algorithms that may be used to find this dominant eigenvector.

Dense Graph

In mathematics, a **dense graph** is a graph in which the number of edges is close to the maximal number of edges. The opposite, a graph with only a few edges, is a **sparse graph**.

The distinction between sparse and dense graphs is rather vague. One possibility is to choose a number k with $1 < k < 2$and to define a *sparse graph* to be a graph with |*E*| = O(|*V*|^k), where |*E*| denotes the number of edges, |*V*| the number of vertices, and the letter O refers to the Big O notation (Preiss 1998, p. 534).

For undirected simple graphs, the **graph density** is defined as:

$$D = \frac{2|E|}{|V|\,(|V|-1)}$$

The maximum number of edges is ½ |*V*| (|*V*|−1), so the maximal density is 1 (for complete graphs) and the minimal density is 0 (Coleman & Moré 1983).

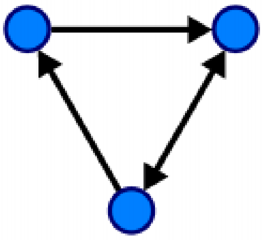
Upper density

Upper density is an extension of the concept of graph density defined above from finite graphs to infinite graphs. Intuitively, an infinite graph has arbitrarily large finite subgraphs with any density less than its upper density, and does not have arbitrarily large finite subgraphs with density greater than its upper density. Formally, the upper density of a graph G is the infimum of the values α such that the finite subgraphs of G with density α have a bounded number of vertices. It can be shown using the Erdős-Stone theorem that the upper density can only be 1 or one of the superparticular ratios 0, 1/2, 2/3, 3/4, 4/5, ... n/(n + 1), ... (see, e.g., Diestel, p. 189).

Sparse and tight graphs

Streinu & Theran (2008) define a graph as being (k,l)-sparse if every nonempty subgraph with n vertices has at most kn − l edges, and (k,l)-tight if it is (k,l)-sparse and has exactly kn − l edges. Thus trees are exactly the (1,1)-tight graphs, forests are exactly the (1,1)-sparse graphs, and graphs with arboricity k are exactly the (k,k)-sparse graphs.
Pseudoforests are exactly the (1,0)-sparse graphs, and the Laman graphs arising in rigidity theory are exactly the (2,3)-tight graphs. Other graph families not characterized by their sparsity can also be described in this way. For instance the facts that any planar graph with n vertices has at most 3n - 6 edges, and that any subgraph of a planar graph is planar, together imply that the planar graphs are (3,6)-sparse. However, not every (3,6)-sparse graph is planar. Similarly, outerplanar graphs are (2,3)-sparse and planar bipartite graphs are (2,4)-sparse. Streinu and Theran show that testing (k,l)-sparsity may be performed in polynomial time

Directed Graphs



A directed graph.

A **directed graph** or **digraph** is a pair $G = (V, A)$ (sometimes $G = (V, E)$) of:[1]

- a set *V*, whose elements are called **vertices** or **nodes,**
- a set *A* of ordered pairs of vertices, called **arcs**, **directed edges**, or **arrows** (and sometimes simply **edges** with the corresponding set named *E* instead of *A*).

It differs from an ordinary or undirected graph, in that the latter is defined in terms of unordered pairs of vertices, which are usually called edges.

Sometimes a digraph is called a **simple digraph** to distinguish it from a **directed multigraph**, in which the arcs constitute a multiset, rather than a set, of ordered pairs of vertices. Also, in a simple digraph loops are disallowed. (A loop is an arc that pairs a vertex to itself.) On the other hand, some texts allow loops, multiple arcs, or both in a digraph.

Basic terminology

An arc $e = (x, y)$ is considered to be directed **from** x **to** y; y is called the **head** and x is called the **tail** of the arc; y is said to be a **direct successor** of x, and x is said to be a **direct predecessor** of y. If a path made up of one or more successive arcs leads from x to y, then y is said to be a **successor** of x, and x is said to be a **predecessor** of y. The arc (y, x) is called the arc (x, y) **inverted**.
A directed graph *G* is called **symmetric** if, for every arc that belongs to *G*, the corresponding inverted arc also belongs to *G*. A symmetric loopless directed graph is equivalent to an undirected graph with the pairs of inverted arcs replaced with edges; thus the number of edges is equal to the number of arcs halved.

The **orientation** of a simple undirected graph is obtained by assigning a direction to each edge. Any directed graph constructed this way is called an **oriented graph**. A distinction between a simple directed graph and an oriented graph is that if x and y are vertices, a simple directed graph allows both (x, y) and (y, x) as edges, while only one is permitted in an oriented graph.[2] [3] [4]

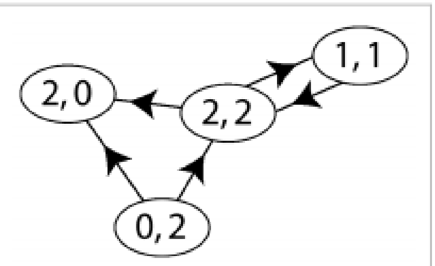
A **weighted digraph** is a digraph with weights assigned for its arcs, similarly to the weighted graph.

The adjacency matrix of a digraph (with loops and multiple arcs) is the integer-valued matrix with rows and columns corresponding to the digraph nodes, where a nondiagonal entry a_{ij} is the number of arcs from node *i* to node *j*, and the diagonal entry a_{ii} is the number of loops at node *i*. The adjacency matrix for a digraph is unique up to the permutations of rows and columns.

Another matrix representation for a digraph is its incidence matrix.

See Glossary of graph theory#Direction for more definitions.

Indegree and outdegree



A digraph with vertices labeled (indegree, outdegree)

For a node, the number of head endpoints adjacent to a node is called the **indegree** of the node and the number of tail endpoints is its **outdegree**.

The indegree is denoted $\deg^-(v)$and the outdegree as $\deg^+(v)$.A vertex with $\deg^-(v)=0$is called a **source**, as it is the origin of each of its incident edges. Similarly, a vertex with $\deg^+(v)=0$is called a **sink**.
The **degree sum formula** states that, for a directed graph

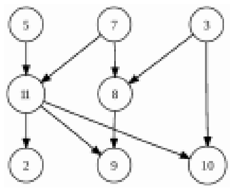
$$\sum_{v\in V}\deg^+(v)=\sum_{v\in V}\deg^-(v)=|A|\,.$$

If for every node, *v* ∈ *V*, $\deg^+(v)=\deg^-(v)$, the graph is called a **balanced digraph**.

Digraph connectivity

A digraph G is called **weakly connected** (or just **connected**[5]) if the undirected **underlying graph** obtained by replacing all directed edges of G with undirected edges is a connected graph. A digraph is **strongly connected** or **strong** if it contains a directed path from *u* to *v* and a directed path from *v* to *u* for every pair of vertices *u*,*v*. The **strong components** are the maximal strongly connected subgraphs.

Classes of digraphs

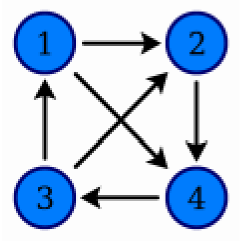


A simple directed acyclic graph

An acyclic digraph (occasionally called a **dag** or **DAG** for "directed acyclic graph", although it is not the same as an orientation of an acyclic graph) is a directed graph with no directed cycles.

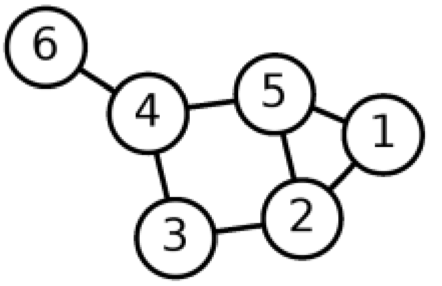
A rooted tree naturally defines an acyclic digraph, if all edges of the underlying tree are directed away from the root.

A tournament is an oriented graph obtained by choosing a direction for each edge in an undirected complete graph.nIn the theory of Lie groups, a quiver Q is a directed graph serving as the domain of, and thus characterizing the shape of, a representation V defined as a functor, specifically an object of the functor category FinVctK F(Q) where F(Q) is the free category on Q consisting of paths in Q and FinVctK is the category of finite dimensional vector spaces over a field K. Representations of a quiver label its vertices with vector spaces and its edges (and hence paths) compatibly with linear transformations between them, and transform via natural transformation.



A tournament on 4 vertices

Vertex(Graph Theory)



A graph with 6 vertices and 7 edges where the vertex no 6 on the far-left is a **leaf vertex** or a **pendant vertex**.

In graph theory, a vertex (plural vertices) or node is the fundamental unit out of which graphs are formed: an undirected graph consists of a set of vertices and a set of edges (unordered pairs of vertices), while a directed graph consists of a set of vertices and a set of arcs (ordered pairs of vertices). From the point of view of graph theory, vertices are treated as featureless and indivisible objects, although they may have additional structure depending on the application from which the graph arises; for instance, a semantic network is a graph in which the vertices represent concepts or classes of objects. The two vertices forming an edge are said to be its endpoints, and the edge is said to be incident to the vertices. A vertex w is said to be adjacent to another vertex v if the graph contains an edge (v,w). The neighborhood of a vertex v is an induced subgraph of the graph, formed by all vertices adjacent to v.

The degree of a vertex in a graph is the number of edges incident to it. An isolated vertex is a vertex with degree zero; that is, a vertex that is not an endpoint of any edge. A leaf vertex (also pendant vertex) is a vertex with degree one. In a directed graph, one can distinguish the outdegree (number of outgoing edges) from the indegree (number of incoming edges); a source vertex is a vertex with indegree zero, while a sink vertex is a vertex with outdegree zero.

A cut vertex is a vertex the removal of which would disconnect the remaining graph; a vertex separator is a collection of vertices the removal of which would disconnect the remaining graph into small pieces. A k-vertex-connected graph is a graph in which removing fewer than k vertices always leaves the remaining graph connected. An independent set is a set of vertices no two of which are adjacent, and a vertex cover is a set of vertices that includes the endpoint of each edge in the graph. The vertex space of a graph is a vector space having a set of basis vectors corresponding with the graph's vertices.

A graph is vertex-transitive if it has symmetries that map any vertex to any other vertex. In the context of graph enumeration and graph isomorphism it is important to distinguish between labeled vertices and unlabeled vertices. A labeled vertex is a vertex that is associated with extra information that enables it to be distinguished from other labeled vertices; two graphs can be considered isomorphic only if the correspondence between their vertices pairs up vertices with equal labels. An unlabeled vertex is one that can be substituted for any other vertex based only on its adjacencies in the graph and not based on any additional information. Vertices in graphs are analogous to, but not the same as, vertices of polyhedra: the skeleton of a polyhedron forms a graph, the vertices of which are the vertices of the polyhedron, but polyhedron vertices have additional structure (their geometric location) that is not assumed to be present in graph theory. The vertex figure of a vertex in a polyhedron is analogous to the neighborhood of a vertex in a graph.

In a directed graph, the forward star of a node u is defined as its outgoing edges. In a Graph G with the set of vertices V and the set of edges E, the forward star of u can be described as

$$\{(u,v) \in E\}.$$ [1]

Flow Network

In graph theory, a flow network is a directed graph where each edge has a capacity and each edge receives a flow. The amount of flow on an edge cannot exceed the capacity of the edge. Often in Operations Research, a directed graph is called a network, the vertices are called nodes and the edges are called arcs. A flow must satisfy the restriction that the amount of flow into a node equals the amount of flow out of it, except when it is a source, which has more outgoing flow, or sink, which has more incoming flow. A network can be used to model traffic in a road system, fluids in pipes, currents in an electrical circuit, or anything similar in which something travels through a network of nodes.

Definition

$G(V,E)$ is a finite directed graph in which every edge $(u,v) \in E$ has a non-negative, real-valued capacity $c(u,v)$. If $(u,v) \notin E$, we assume that $c(u,v) = 0$. We distinguish two vertices: a source s and a sink t. A flow network is a real function $f : V \times V \to \mathbb{R}$ with the following three properties for all nodes u and v:

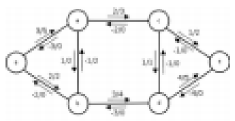
Capacity constraints:	$f(u,v) \le c(u,v)$. The flow along an edge cannot exceed its capacity.
Skew symmetry:	$f(u,v) = -f(v,u)$. The net flow from u to v must be the opposite of the net flow from v to u (see example).
Flow conservation:	$\sum_{w \in V} f(u,w) = 0$, unless $u = s$ or $w = t$. The net flow to a node is zero, except for the source, which "produces" flow, and the sink, which "consumes" flow.

Notice that $f(u,v)$ is the *net* flow from u to v. If the graph represents a physical network, and if there is a real flow of, for example, 4 units from u to v, and a real flow of 3 units from v to u, we have $f(u,v) = 1$ and $f(v,u) = -1$.

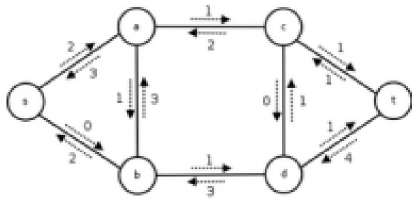
The **residual capacity** of an edge is $c_f(u,v) = c(u,v) - f(u,v)$. This defines a **residual network** denoted $G_f(V,E_f)$, giving the amount of *available* capacity. See that there can be an edge from u to v in the residual network, even though there is no edge from u to v in the original network. Since flows in opposite directions cancel out, *decreasing* the flow from v to u is the same as *increasing* the flow from u to v. An **augmenting path** is a path $(u_1, u_2, \ldots, u_k)$ in the residual network, where $u_1 = s$, $u_k = t$, and $c_f(u_i, u_{i+1}) > 0$. A network is at maximum flow if and only if there is no augmenting path in the residual network.

Example:

To the right you see a flow network with source labeled s, sink t, and four additional nodes. The flow and capacity is denoted f/c. Notice how the network upholds skew symmetry, capacity constraints and flow conservation. The total amount of flow from s to t is 5, which can be easily seen from the fact that the total outgoing flow from s is 5, which is also the incoming flow to t. We know that no flow appears or disappears in any of the other nodes.



A flow network showing flow and capacity



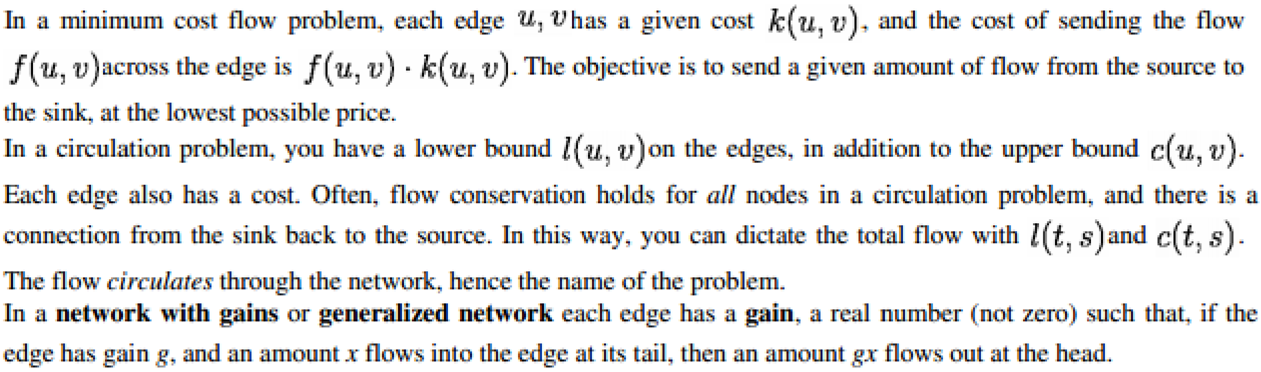
Residual network for the above flow network, showing residual capacities

Below you see the residual network for the given flow. Notice how there is positive residual capacity on some edges where the original capacity is zero, for example for the edge (d,c). This flow is not a maximum flow. There is available capacity along the paths (s,a,c,t), $(s,a,b,d,t)$and (s,a,b,d,c,t), which are then the augmenting paths. The residual capacity of the first path is $\min(c(s,a)-f(s,a),c(a,c)-f(a,c),c(c,t)-f(c,t))$ $=\min(5-3,3-2,2-1)=\min(2,1,1)=1$. Notice that augmenting path (s,a,b,d,c,t)does not exist in the original network, but you can send flow along it, and still get a legal flow.
If this is a real network, there might actually be a flow of 2 from a to b, and a flow of 1 from b to a, but we only maintain the **net** flow.

Applications:

Picture a series of water pipes, fitting into a network. Each pipe is of a certain diameter, so it can only maintain a flow of a certain amount of water. Anywhere that pipes meet, the total amount of water coming into that junction must be equal to the amount going out, otherwise we would quickly run out of water, or we would have a build up of water. We have a water inlet, which is the source, and an outlet, the sink. A flow would then be one possible way for water to get from source to sink so that the total amount of water coming out of the outlet is consistent. Intuitively, the total flow of a network is the rate at which water comes out of the outlet. Flows can pertain to people or material over transportation networks, or to electricity over electrical distribution systems. For any such physical network, the flow coming into any intermediate node needs to equal the flow going out of that node. Bollobás characterizes this constraint in terms of Kirchhoff's current law, while later authors (i.e.: Chartrand) mention its generalization to some conservation equation.
Flow networks also find applications in ecology: flow networks arise naturally when considering the flow of nutrients and energy between different organizations in a food web. The mathematical problems associated with such networks are quite different from those that arise in networks of fluid or traffic flow. The field of ecosystem network analysis, developed by Robert Ulanowicz and others, involves using concepts from information theory and thermodynamics to study the evolution of these networks over time. The simplest and most common problem using flow networks is to find what is called the maximum flow, which

provides the largest possible total flow from the source to the sink in a given graph. There are many other problems which can be solved using max flow algorithms, if they are appropriately modeled as flow networks, such as bipartite matching, the assignment problem and the transportation problem. In a multi-commodity flow problem, you have multiple sources and sinks, and various "commodities" which are to flow from a given source to a given sink. This could be for example various goods that are produced at various factories, and are to be delivered to various given customers through the same transportation network.



In a minimum cost flow problem, each edge u, v has a given cost $k(u, v)$, and the cost of sending the flow $f(u, v)$ across the edge is $f(u, v) \cdot k(u, v)$. The objective is to send a given amount of flow from the source to the sink, at the lowest possible price.

In a circulation problem, you have a lower bound $l(u, v)$ on the edges, in addition to the upper bound $c(u, v)$. Each edge also has a cost. Often, flow conservation holds for *all* nodes in a circulation problem, and there is a connection from the sink back to the source. In this way, you can dictate the total flow with $l(t, s)$ and $c(t, s)$. The flow *circulates* through the network, hence the name of the problem.

In a **network with gains** or **generalized network** each edge has a **gain**, a real number (not zero) such that, if the edge has gain *g*, and an amount *x* flows into the edge at its tail, then an amount *gx* flows out at the head.

Cycle(Graph Theory)

In graph theory, the term cycle may refer to several closely related objects.

- A closed walk, with repeated vertices allowed. See path (graph theory). (This usage is common in computer science. In graph theory it is more often called a closed walk.)
- A closed (simple) path, with no other repeated vertices or edges other than the starting and ending vertices. (This usage is common in graph theory, see "Cycle graph") This may also be called a simple cycle, circuit, circle, or polygon.
- A closed directed walk, with repeated vertices allowed. (This usage is common in computer science. In graph theory it is more often called a closed directed walk.)
- A closed directed (simple) path, with no repeated vertices other than the starting and ending vertices. (This usage is common in graph theory.) This may also be called a simple (directed) cycle.
- The edge set of an undirected closed path without repeated vertices or edges. This may also be called a circuit, circle, or polygon.
- An element of the binary or integral (or real, complex, etc.) cycle space of a graph. (This is the usage closest to that in the rest of mathematics, in particular algebraic topology.) Such a cycle may be called a binary cycle, integral cycle, etc.
- An edge set which has even degree at every vertex; also called an even edge set or, when taken together with its vertices, an even subgraph. This is equivalent to a binary cycle, since a binary cycle is the indicator function of an edge set of this type.

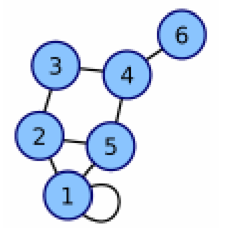
Chordless cycles in a graph are sometimes called graph holes. A graph antihole is the complement of a graph hole.

Adjacency Matrix

In mathematics and computer science, an adjacency matrix is a means of representing which vertices of a graph are adjacent to which other vertices. Another matrix representation for a graph is the incidence matrix.
Specifically, the adjacency matrix of a finite graph G on n vertices is the n × n matrix where the non-diagonal entry aij is the number of edges from vertex i to vertex j, and the diagonal entry aii, depending on the convention, is either once or twice the number of edges (loops) from vertex i to itself. Undirected graphs often use the former convention
of counting loops twice, whereas directed graphs typically use the latter convention. There exists a unique adjacency matrix for each isomorphism class of graphs (up to permuting rows and columns), and it is not the adjacency matrix of any other isomorphism class of graphs. In the special case of a finite simple graph, the adjacency matrix is a (0,1)-matrix with zeros on its diagonal. If the graph is undirected, the adjacency matrix is symmetric.
The relationship between a graph and the eigenvalues and eigenvectors of its adjacency matrix is studied in spectral graph theory.

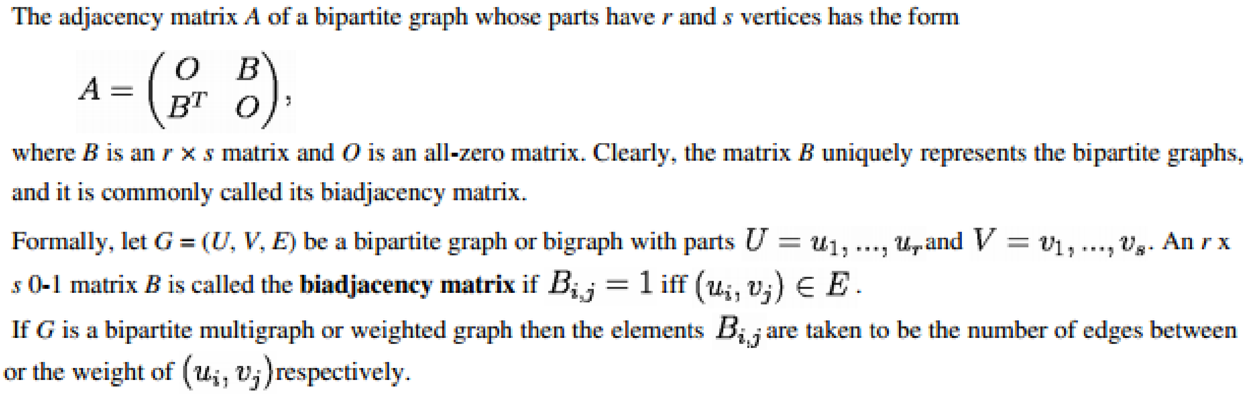
Examples:

• Here is an example of a labeled graph and its adjacency matrix. The convention followed here is that an adjacent edge counts 1 in the matrix for an undirected graph. (X,Y coordinates are 1-6)

Labeled graph	Adjacency matrix
	$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$

• The adjacency matrix of a complete graph is all 1's except for 0's on the diagonal.
• The adjacency matrix of an empty graph is a zero matrix.

Adjacency Matrix of a Bi-Partite Graph



The adjacency matrix *A* of a bipartite graph whose parts have *r* and *s* vertices has the form

$$A = \begin{pmatrix} O & B \\ B^T & O \end{pmatrix},$$

where *B* is an *r* × *s* matrix and *O* is an all-zero matrix. Clearly, the matrix *B* uniquely represents the bipartite graphs, and it is commonly called its biadjacency matrix.

Formally, let *G* = (*U*, *V*, *E*) be a bipartite graph or bigraph with parts $U = u_1, ..., u_r$and $V = v_1, ..., v_s$. An *r* x *s* 0-1 matrix *B* is called the **biadjacency matrix** if $B_{i,j} = 1$ iff $(u_i, v_j) \in E$.

If *G* is a bipartite multigraph or weighted graph then the elements $B_{i,j}$ are taken to be the number of edges between or the weight of (u_i, v_j)respectively.

Properties

The adjacency matrix of an undirected simple graph is symmetric, and therefore has a complete set of real eigenvalues and an orthogonal eigenvector basis. The set of eigenvalues of a graph is the **spectrum** of the graph.

Suppose two directed or undirected graphs $G_1$and $G_2$with adjacency matrices $A_1$and $A_2$are given. $G_1$and $G_2$are isomorphic if and only if there exists a permutation matrix Psuch that

$$PA_1P^{-1} = A_2.$$

In particular, $A_1$and $A_2$are similar and therefore have the same minimal polynomial, characteristic polynomial, eigenvalues, determinant and trace. These can therefore serve as isomorphism invariants of graphs. However, two graphs may possess the same set of eigenvalues but not be isomorphic.

If *A* is the adjacency matrix of the directed or undirected graph *G*, then the matrix A^n (i.e., the matrix product of *n* copies of *A*) has an interesting interpretation: the entry in row *i* and column *j* gives the number of (directed or undirected) walks of length *n* from vertex *i* to vertex *j*. This implies, for example, that the number of triangles in an undirected graph *G* is exactly the trace of A^3 divided by 6.

The main diagonal of every adjacency matrix corresponding to a graph without loops has all zero entries.

For (d)-regular graphs, d is also an eigenvalue of A for the vector $v = (1, \dots, 1)$, and G is connected if and only if the multiplicity of d is 1. It can be shown that $-d$ is also an eigenvalue of A if G is a connected bipartite graph. The above are results of Perron–Frobenius theorem.

Variations

The Seidel adjacency matrix or **(0,−1,1)-adjacency matrix** of a simple graph has zero on the diagonal and entry $a_{ij} = -1$if *ij* is an edge and +1 if it is not. This matrix is used in studying strongly regular graphs and two-graphs.

A distance matrix is like a higher-level adjacency matrix. Instead of only providing information about whether or not two vertices are connected, also tells the distances between them. This assumes the length of every edge is 1. A variation is where the length of an edge is not necessarily 1.

Data Structures

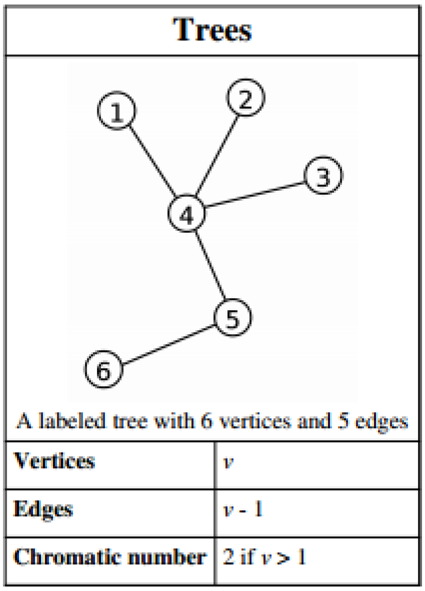
When used as a data structure, the main alternative for the adjacency matrix is the adjacency list. Because each entry in the adjacency matrix requires only one bit, they can be represented in a very compact way, occupying only $n^2/8$ bytes of contiguous space, where n is the number of vertices. Besides just avoiding wasted space, this compactness encourages locality of reference.

On the other hand, for a sparse graph, adjacency lists win out, because they do not use any space to represent edges which are *not* present. Using a naïve array implementation on a 32-bit computer, an adjacency list for an undirected graph requires about $8e$ bytes of storage, where e is the number of edges.

Noting that a simple graph can have at most n^2edges, allowing loops, we can let $d = e/n^2$denote the *density* of the graph. Then, $8e > n^2/8$, or the adjacency list representation occupies more space, precisely when $d > 1/64$. Thus a graph must be sparse indeed to justify an adjacency list representation.

Besides the space tradeoff, the different data structures also facilitate different operations. Finding all vertices adjacent to a given vertex in an adjacency list is as simple as reading the list. With an adjacency matrix, an entire row must instead be scanned, which takes O(n) time. Whether there is an edge between two given vertices can be determined at once with an adjacency matrix, while requiring time proportional to the minimum degree of the two vertices with the adjacency list.

Tree (Graph Theory)



Trees

A labeled tree with 6 vertices and 5 edges

Vertices	*v*
Edges	*v* - 1
Chromatic number	2 if *v* > 1

In mathematics, more specifically graph theory, a tree is an undirected graph in which any two vertices are connected by exactly one simple path. In other words, any connected graph without cycles is a tree. A forest is a disjoint union of trees.

The various kinds of data structures referred to as trees in computer science are similar to trees in graph theory, except that computer science trees have directed edges. Although they do not meet the definition given here, these graphs are referred to in graph theory as ordered directed trees (see below).

Definitions

A tree is an undirected simple graph G that satisfies any of the following equivalent conditions:

- G is connected and has no cycles.
- G has no cycles, and a simple cycle is formed if any edge is added to G.
- G is connected, and it is not connected anymore if any edge is removed from G.
- G is connected and the 3-vertex complete graph is not a minor of G.
- Any two vertices in G can be connected by a unique simple path.

If G has finitely many vertices, say n of them, then the above statements are also equivalent to any of the following conditions:

- G is connected and has n − 1 edges.
- G has no simple cycles and has n − 1 edges.

An irreducible (or series-reduced) tree is a tree in which there is no vertex of degree 2.

A forest is an undirected graph, all of whose connected components are trees; in other words, the graph consists of a disjoint union of trees. Equivalently, a forest is an undirected cycle-free graph. As special cases, an empty graph, a single tree, and the discrete graph on a set of vertices (that is, the graph with these vertices that has no edges), all are examples of forests.

The term hedge sometimes refers to an ordered sequence of trees.

A polytree or oriented tree is a directed graph with at most one undirected path between any two vertices. In other words, a polytree is a directed acyclic graph for which there are no undirected cycles either.

A directed tree is a directed graph which would be a tree if the directions on the edges were ignored. Some authors restrict the phrase to the case where the edges are all directed towards a particular vertex, or all directed away from a particular vertex (see arborescence).
A tree is called a rooted tree if one vertex has been designated the root, in which case the edges have a natural orientation, towards or away from the root. The tree-order is the partial ordering on the vertices of a tree with $u \leq v$ if and only if the unique path from the root to v passes through u. A rooted tree which is a subgraph of some graph G is a normal tree if the ends of every edge in G are comparable in this tree-order whenever those ends are vertices of the tree (Diestel 2005, p. 15). Rooted trees, often with additional structure such as ordering of the neighbors at each vertex, are a key data structure in computer science; see tree data structure. In a context where trees are supposed to have a root, a tree without any designated root is called a free tree.
In a rooted tree, the parent of a vertex is the vertex connected to it on the path to the root; every vertex except the root has a unique parent. A child of a vertex v is a vertex of which v is the parent. A leaf is a vertex without children.
A labeled tree is a tree in which each vertex is given a unique label. The vertices of a labeled tree on n vertices are typically given the labels 1, 2, …, n. A recursive tree is a labeled rooted tree where the vertex labels respect the tree order (i.e., if $u < v$ for two vertices u and v, then the label of u is smaller than the label of v).
An ordered tree is a rooted tree for which an ordering is specified for the children of each vertex.
An n-ary tree is a rooted tree for which each vertex which is not a leaf has at most n children. 2-ary trees are
sometimes called binary trees, while 3-ary trees are sometimes called ternary trees.
A terminal vertex of a tree is a vertex of degree 1. In a rooted tree, the leaves are all terminal vertices; additionally, the root, if not a leaf itself, is a terminal vertex if it has precisely one child.

Example

The example tree shown to the right has 6 vertices and $6 − 1 = 5$ edges. The unique simple path connecting the vertices 2 and 6 is 2-4-5-6.

Facts

- Every tree is a bipartite graph and a median graph. Every tree with only countably many vertices is a planar graph.
- Every connected graph G admits a spanning tree, which is a tree that contains every vertex of G and whose edges are edges of G.
- Every connected graph with only countably many vertices admits a normal spanning tree (Diestel 2005, Prop. 8.2.4).
- There exist connected graphs with uncountably many vertices which do not admit a normal spanning tree (Diestel 2005, Prop. 8.5.2).
- Every finite tree with n vertices, with $n > 1$, has at least two terminal vertices. This minimal number of terminal vertices is characteristic of path graphs; the maximal number, $n − 1$, is attained by star graphs.
- For any three vertices in a tree, the three paths between them have exactly one vertex in common.

Enumeration

Given n labeled vertices, there are n^{n-2} different ways to connect them to make a tree. This result is called Cayley's formula. It can be proven by first showing that the number of trees with n vertices of degree $d_1,d_2,...,d_n$ is the multinomial coefficient

$$\binom{n-2}{d_1-1, d_2-1, \ldots, d_n-1}.$$

An alternative proof uses Prüfer sequences. This is the special case for complete graphs of a more general problem, counting the number of spanning trees in an undirected graph, which can be achieved by computing a determinant according to the matrix tree theorem. The similar problem of counting all the subtrees regardless of size has been shown to be #P-complete in the general case (Jerrum (1994)).

Counting the number of unlabeled free trees is a harder problem. No closed formula for the number $t(n)$ of trees with n vertices up to graph isomorphism is known. The first few values of $t(n)$ are 1, 1, 1, 1, 2, 3, 6, 11, 23, 47, 106, 235, 551, 1301, 3159, ... (sequence A000055 [1] in OEIS). Otter (1948) proved that

$$t(n) \sim C\alpha^n n^{-5/2} \quad \text{as } n \to \infty,$$

with $C = 0.534949606\ldots$ and $\alpha = 2.99557658565\ldots$ (Here, $f \sim g$means that $\lim_{n\to\infty} f/g = 1$, where $f = t(n)$ and $g = C\alpha^n n^{-5/2}$). Similarly, he showed that for the number $r(n)$of unlabeled rooted trees with n vertices holds

$$r(n) \sim D\alpha^n n^{-5/2} \quad \text{as } n \to \infty,$$

with $D = 0.43992401257\ldots$ and α the same as above (cf. Knuth (1997), Chap. 2.3.4.4 and Flajolet & Sedgewick (2009), Chap. VII.5).

Types of trees

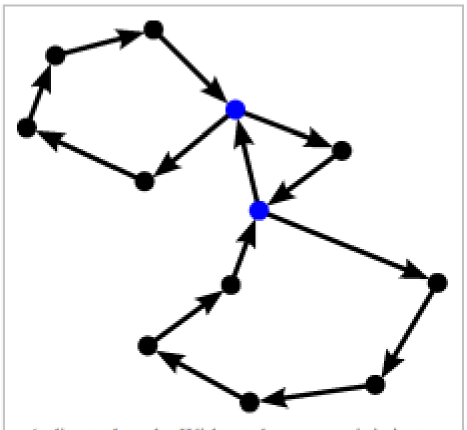
A star is a tree in which there is only one internal node and n − 1 leaves; that is, a star is a tree with as many leaves as possible. A tree with two leaves, the fewest possible, is a path graph. If all nodes in a tree are within distance one of a central path, then the tree is a caterpillar tree. If all nodes are within distance two of a central path, then the tree is a lobster.

Path(Graph Theory)

In graph theory, a path in a graph is a sequence of vertices such that from each of its vertices there is an edge to the next vertex in the sequence. A path may be infinite, but a finite path always has a first vertex, called its start vertex, and a last vertex, called its end vertex. Both of them are called end or terminal vertices of the path. The other vertices in the path are internal vertices. A cycle is a path such that the start vertex and end vertex are the same. Note that the choice of the start vertex in a cycle is arbitrary.

Paths and cycles are fundamental concepts of graph theory, described in the introductory sections of most graph theory texts. See e.g. Bondy and Murty (1976), Gibbons (1985), or Diestel (2005). Korte et al. (1990) cover more advanced algorithmic topics concerning paths in graphs.

Different types of path


A directed cycle. Without the arrows, it is just a cycle. This is not a simple cycle, since the blue vertices are used twice.

The same concepts apply both to undirected graphs and directed graphs, with the edges being directed from each vertex to the following one. Often the terms directed path and directed cycle are used in the directed case.
A path with no repeated vertices is called a simple path, and a cycle with no repeated vertices or edges aside from the necessary repetition of the start and end vertex is a simple cycle. In modern graph theory, most often "simple" is implied; i.e., "cycle" means "simple cycle" and "path" means "simple path", but this convention is not always observed, especially in applied graph theory. Some authors (e.g. Bondy and Murty 1976) use the term "walk" for a path in which vertices or edges may be repeated, and reserve the term "path" for what is here called a simple path. A path such that no graph edges connect two nonconsecutive path vertices is called an induced path. A simple cycle that includes every vertex, without repetition, of the graph is known as a Hamiltonian cycle.
A cycle with just one edge removed in the corresponding spanning tree of the original graph is known as a Fundamental cycle.
Two paths are independent (alternatively, internally vertex-disjoint) if they do not have any internal vertex in common.
The length of a path is the number of edges that the path uses, counting multiple edges multiple times. The length can be zero for the case of a single vertex.
A weighted graph associates a value (weight) with every edge in the graph. The weight of a path in a weighted graph is the sum of the weights of the traversed edges. Sometimes the words cost or length are used instead of weight.

Glossary of Graph Theory

Graph theory is a growing area in mathematical research, and has a large specialized vocabulary. Some authors use the same word with different meanings. Some authors use different words to mean the same thing. This page attempts to keep up with current usage.

Basics

A graph G consists of two types of elements, namely vertices and edges. Every edge has two endpoints in the set of vertices, and is said to connect or join the two endpoints. An edge can thus be defined as a set of two vertices (or an ordered pair, in the case of a directed graph - see Section Direction).
Alternative models of graphs exist; e.g., a graph may be thought of as a Boolean binary function over the set of vertices or as a square (0,1)-matrix.
A vertex is simply drawn as a node or a dot. The vertex set of G is usually denoted by V(G), or V when there is no danger of confusion. The order of a graph is the number of its vertices, i.e. |V(G)|.
An edge (a set of two elements) is drawn as a line connecting two vertices, called endpoint or (less often) end vertices. An edge with endvertices x and y is denoted by xy (without any

symbol in between). The edge set of G is usually denoted by E(G), or E when there is no danger of confusion.

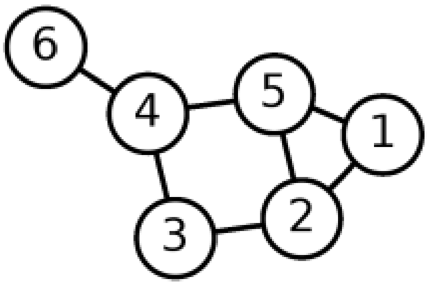
The size of a graph is the number of its edges, i.e. |E(G)|.

A loop is an edge whose endpoints are the same vertex. A link has two distinct endvertices. An edge is multiple if there is another edge with the same endvertices; otherwise it is simple. The multiplicity of an edge is the number of multiple edges sharing the same endvertices; the multiplicity of a graph, the maximum multiplicity of its edges. A graph is a simple graph if it has no multiple edges or loops, a multigraph if it has multiple edges, but no loops, and a multigraph or pseudograph if it contains both multiple edges and loops (the literature is highly inconsistent).

When stated without any qualification, a graph is almost always assumed to be simple—one has to judge from the context.

Graphs whose edges or vertices have names or labels are known as labeled, those without as unlabeled. Graphs with labeled vertices only are vertex-labeled, those with labeled edges only are edge-labeled. The difference between a labeled and an unlabeled graph is that the latter has no specific set of vertices or edges; it is regarded as another way to look upon an isomorphism type of graphs. (Thus, this usage distinguishes between graphs with identifiable vertex or edge sets on the one hand, and isomorphism types or classes of graphs on the other.) (Graph labeling usually refers to the assignment of labels (usually natural numbers, usually distinct) to the edges and vertices of a graph, subject to certain rules depending on the situation. This should not be confused with a graph's merely having distinct labels or names on the vertices.)

A hyperedge is an edge that is allowed to take on any number of vertices, possibly more than 2. A graph that allows any hyperedge is called a hypergraph. A simple graph can be considered a special case of the hypergraph, namely the 2-uniform hypergraph. However, when stated without any qualification, an edge is always assumed to consist of at most 2 vertices, and a graph is never confused with a hypergraph.



A labeled simple graph with vertex set $V = \{1, 2, 3, 4, 5, 6\}$ and edge set $E =$.

A **non-edge** (or **anti-edge**) is an edge that is not present in the graph. More formally, for two vertices u and v, $\{u, v\}$ is a non-edge in a graph G whenever $\{u, v\}$ is not an edge in G. This means that there is either no edge between the two vertices or (for directed graphs) at most one of (u, v) and (v, u) from v is an arc in G.

Occasionally the term **cotriangle** or **anti-triangle** is used for a set of three vertices none of which are connected.

The **complement** $\bar{G}$ of a graph G is a graph with the same vertex set as G but with an edge set such that xy is an edge in $\bar{G}$ if and only if xy is not an edge in G.

An **edgeless graph** or **empty graph** or **null graph** is a graph with zero or more vertices, but no edges. The **empty graph** or **null graph** may also be the graph with no vertices and no edges. If it is a graph with no edges and any number n of vertices, it may be called the **null graph on** n **vertices**. (There is no consistency at all in the literature.)

A graph is infinite if it has infinitely many vertices or edges or both; otherwise the graph is finite. An infinite graph where every vertex has finite degree is called locally finite. When stated without any qualification, a graph is usually assumed to be finite. See also continuous graph.

Two graphs G and H are said to be isomorphic, denoted by G ~ H, if there is a one-to-one correspondence, called an isomorphism, between the vertices of the graph such that two vertices are adjacent in G if and only if their corresponding vertices are adjacent in H. Likewise, a graph G is said to be homomorphic to a graph H if there is a mapping, called a homomorphism, from V(G) to V(H) such that if two vertices are adjacent in G then their corresponding vertices are adjacent in H.

Subgraphs

A subgraph of a graph G is a graph whose vertex set is a subset of that of G, and whose adjacency relation is a subset of that of G restricted to this subset. In the other direction, a supergraph of a graph G is a graph of which G is a subgraph. We say a graph G contains another graph H if some subgraph of G is H or is isomorphic to H.

A subgraph H is a spanning subgraph, or factor, of a graph G if it has the same vertex set as G. We say H spans G.

A subgraph H of a graph G is said to be induced if, for any pair of vertices x and y of H, xy is an edge of H if and only if xy is an edge of G. In other words, H is an induced subgraph of G if it has exactly the edges that appear in G over the same vertex set. If the vertex set of H is the subset S of V(G), then H can be written as G[S] and is said to be induced by S.

A graph that does not contain H as an induced subgraph is said to be H-free.

A universal graph in a class K of graphs is a simple graph in which every element in K can be embedded as a subgraph.

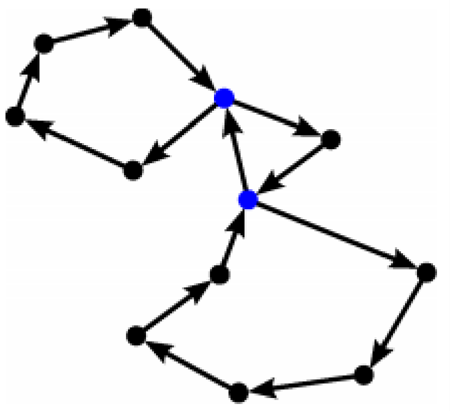
Walks

A walk is an alternating sequence of vertices and edges, beginning and ending with a vertex, where each vertex is incident to both the edge that precedes it and the edge that follows it in the sequence, and where the vertices that precede and follow an edge are the end vertices of that edge. A walk is closed if its first and last vertices are the same, and open if they are different.

The length l of a walk is the number of edges that it uses. For an open walk, l = n–1, where n is the number of vertices visited (a vertex is counted each time it is visited). For a closed walk, l = n (the start/end vertex is listed twice, but is not counted twice). In the example graph, (1, 2, 5, 1, 2, 3) is an open walk with length 5, and (4, 5, 2, 1,5, 4) is a closed walk of length 5.

A trail is a walk in which all the edges are distinct. A closed trail has been called a tour or circuit, but these are notuniversal, and the latter is often reserved for a regular subgraph of degree two.

Traditionally, a path referred to what is now usually known as an open walk. Nowadays, when stated without any qualification, a path is usually understood to be simple, meaning that no vertices (and thus no edges) are repeated. (The term chain has also been used to refer to a walk in which all vertices and edges are



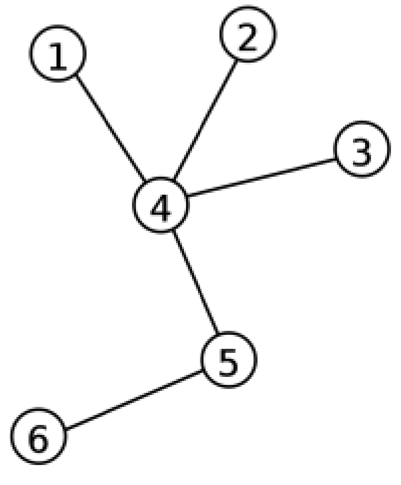
A directed cycle. Without the arrows, it is just a cycle. This is not a simple cycle, since the blue vertices are used twice.

distinct.) In the example graph, (5, 2, 1) is a path of length 2. The closed equivalent to this type of walk, a walk that starts and ends at the same vertex but otherwise has no repeated vertices or edges, is called a cycle. Like path, this term traditionally referred to any closed walk, but now is usually understood to be simple by definition. In the example graph, (1, 5, 2, 1) is a cycle of length 3. (A cycle, unlike a path, is not allowed to have length 0.)
Paths and cycles of n vertices are often denoted by Pn and Cn, respectively. (Some authors use the length instead of the number of vertices, however.)

C1 is a loop, C2 is a digon (a pair of parallel undirected edges in a multigraph, or a pair of antiparallel edges in a directed graph), and C3 is called a triangle.
A cycle that has odd length is an odd cycle; otherwise it is an even cycle. One theorem is that a graph is bipartite if and only if it contains no odd cycles. (See complete bipartite graph.)
A graph is acyclic if it contains no cycles; unicyclic if it contains exactly one cycle; and pancyclic if it contains cycles of every possible length (from 3 to the order of the graph).
The girth of a graph is the length of a shortest (simple) cycle in the graph; and the circumference, the length of a longest (simple) cycle. The girth and circumference of an acyclic graph are defined to be infinity ∞.
A path or cycle is Hamiltonian (or spanning) if it uses all vertices exactly once. A graph that contains a Hamiltonian path is traceable; and one that contains a Hamiltonian path for any given pair of (distinct) end vertices is a Hamiltonian connected graph. A graph that contains a Hamiltonian cycle is a Hamiltonian graph.
A trail or circuit (or cycle) is Eulerian if it uses all edges precisely once. A graph that contains an Eulerian trail is traversable. A graph that contains an Eulerian circuit is an Eulerian graph. Two paths are internally disjoint (some people call it independent) if they do not have any vertex in common, except the first and last ones.
A theta graph is the union of three internally disjoint (simple) paths that have the same two distinct end vertices. A theta0 graph has seven vertices which can be arranged as the vertices of a regular hexagon plus an additional vertex in the center. The eight edges are the perimeter of the hexagon plus one diameter.

Trees

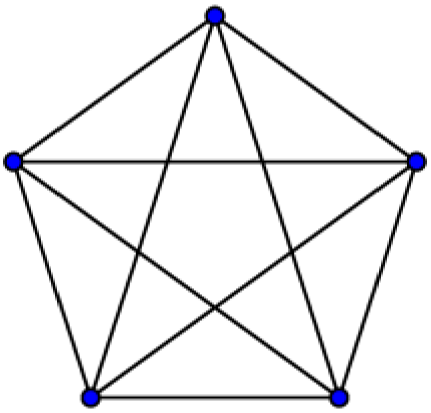
A tree is a connected acyclic simple graph. A vertex of degree 1 is called a leaf, or pendant vertex. An edge incident to a leaf is a leaf edge, or pendant edge. (Some people define a leaf edge as a leaf and then define a leaf vertex on top of it. These two sets of definitions are often used interchangeably.)
A non-leaf vertex is an internal vertex.
Sometimes, one vertex of the tree is distinguished, and called the root; in this case, the tree is called rooted. Rooted trees are often treated as directed acyclic graphs with the edges pointing away from the root.
A subtree of the tree T is a connected subgraph of T.
A forest is an acyclic simple graph.
A subforest of the forest F is a subgraph of F.
A spanning tree is a spanning subgraph that is a tree.
Every graph has a spanning forest. But only a connected graph has a spanning tree.
A special kind of tree called a star is K1,k .



A labeled tree with 6 vertices and 5 edges.

An induced star with 3 edges is a claw.
A caterpillar is a tree in which all non-leaf nodes form a single path.
A k-ary tree is a rooted tree in which every internal vertex has k children. A 1-ary tree is just a path. A 2-ary tree is also called a binary tree.

Cliques



K_5, a complete graph. If a subgraph looks like this, the vertices in that subgraph form a clique of size 5.

The complete graph K
n of order n is a simple graph with n vertices in which every vertex is adjacent to every other. The example graph to the right is complete. The complete graph on n vertices is often
denoted by K n. It has n(n-1)/2 edges (corresponding to all possible
choices of pairs of vertices).
A clique in a graph is a set of pairwise adjacent vertices. Since any subgraph induced by a clique is a complete subgraph, the two terms
and their notations are usually used interchangeably.
A k-clique is a clique of order k. In the example graph above, vertices 1, 2 and 5 form a 3-clique, or a triangle. A maximal clique is a clique that is not a subset of any other clique (some authors reserve the term clique for maximal cliques).
The clique number ω(G) of a graph G is the order of a largest clique in G.

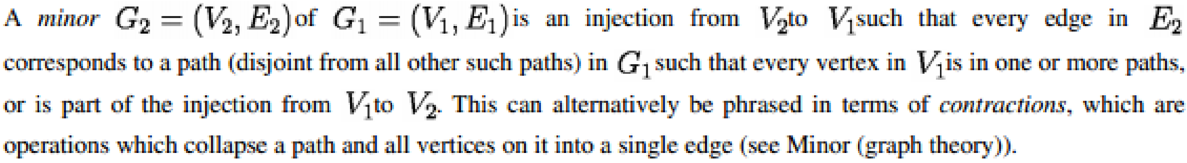
Strongly connected component

A related but weaker concept is that of a strongly connected component. Informally, a strongly connected component of a directed graph is a subgraph where all nodes in the subgraph are reachable by all other nodes in the subgraph.
Reachability between nodes is established by the existence of a path between the nodes.
A directed graph can be decomposed into strongly connected components by running the depth-first search (DFS) algorithm twice: first, on the graph itself and next on the transpose of the graph in decreasing order of the finishing times of the first DFS. Given a directed graph G, the transpose GT is the graph G with all the edge directions reversed.

Knots

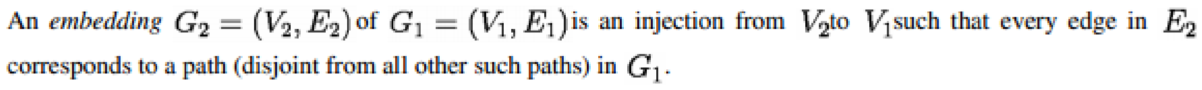
A knot in a directed graph is a collection of vertices and edges with the property that every vertex in the knot has outgoing edges, and all outgoing edges from vertices in the knot terminate at other vertices in the knot. Thus it is impossible to leave the knot while following the directions of the edges.

Minors



A *minor* $G_2 = (V_2, E_2)$ of $G_1 = (V_1, E_1)$ is an injection from V_2 to V_1 such that every edge in E_2 corresponds to a path (disjoint from all other such paths) in G_1 such that every vertex in V_1 is in one or more paths, or is part of the injection from V_1 to V_2. This can alternatively be phrased in terms of *contractions*, which are operations which collapse a path and all vertices on it into a single edge (see Minor (graph theory)).

Embedding


An *embedding* $G_2 = (V_2, E_2)$ of $G_1 = (V_1, E_1)$ is an injection from $V_2$to V_1such that every edge in E_2 corresponds to a path (disjoint from all other such paths) in G_1.

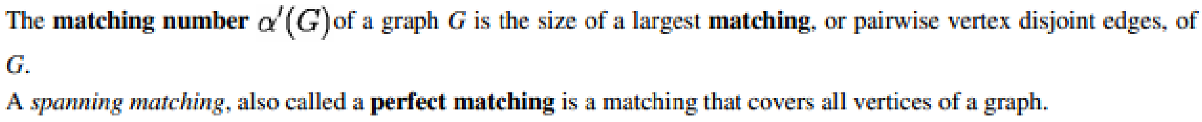
Adjacency and degree

In graph theory, degree, especially that of a vertex, is usually a measure of immediate adjacency.
An edge connects two vertices; these two vertices are said to be incident to that edge, or, equivalently, that edge incident to those two vertices. All degree-related concepts have to do with adjacency or incidence.
The degree, or valency, d G(v) of a vertex v in a graph G is the number of edges incident to v, with loops being counted twice. A vertex of degree 0 is an isolated vertex. A vertex of degree 1 is a leaf. In the labelled simple graph example, vertices 1 and 3 have a degree of 2, vertices 2, 4 and 5 have a degree of 3, and vertex 6 has a degree of 1. If E is finite, then the total sum of vertex degrees is equal to twice the number of edges.
The total degree of a graph is equal to two times the number of edges, loops included. This means that for a graph with 3 vertices with each vertex having a degree of two (i.e. a triangle) the total degree would be six (e.g. 3 x 2 = 6).
The general formula for this is total degree = 2n where n = number of edges.
A degree sequence is a list of degrees of a graph in non-increasing order (e.g. d 1 ≥ d2 ≥ … ≥ dn). A sequence of non-increasing integers is realizable if it is a degree sequence of some graph.
Two vertices u and v are called adjacent if an edge exists between them. We denote this by u ~ v or u ↓ v. In the above graph, vertices 1 and 2 are adjacent, but vertices 2 and 4 are not. The set of neighbors of v, that is, vertices adjacent to v not including v itself, forms an induced subgraph called the (open) neighborhood of v and denoted N G(v). When v is also included, it is called a closed neighborhood and denoted by N G[v]. When stated without any qualification, a neighborhood is assumed to be open. The subscript G is usually dropped when there is no danger of confusion; the same neighborhood notation may also be used to refer to sets of adjacent vertices rather than the corresponding induced subgraphs. In the example graph, vertex 1 has two neighbors: vertices 2 and 5. For a simple graph, the number of neighbors that a vertex has coincides with its degree.
A dominating set of a graph is a vertex subset whose closed neighborhood includes all vertices of the graph. A vertex v dominates another vertex u if there is an edge from v to u. A vertex subset V dominates another vertex subset U if every vertex in U is adjacent to some vertex in V. The minimum size of a dominating set is thedomination number γ(G).
In computers, a finite, directed or undirected graph (with n vertices, say) is often represented by its adjacency matrix: an n-by-n matrix whose entry in row i and column j gives the number of edges from the i-th to the j-th vertex.
Spectral graph theory studies relationships between the properties of the graph and its adjacency matrix.
The maximum degree Δ(G) of a graph G is the largest degree over all vertices; the minimum degree δ(G), thesmallest.
A graph in which every vertex has the same degree is regular. It is k-regular if every vertex has degree k. A
0-regular graph is an independent set. A 1-regular graph is a matching. A 2-regular graph is a vertex disjoint union of cycles. A 3-regular graph is said to be cubic, or trivalent.

A k-factor is a k-regular spanning subgraph. A 1-factor is a perfect matching. A partition of edges of a graph into k-factors is called a k-factorization. A k-factorable graph is a graph that admits a k-factorization.
A graph is biregular if it has unequal maximum and minimum degrees and every vertex has one of those two degrees.
A strongly regular graph is a regular graph such that any adjacent vertices have the same number of common neighbors as other adjacent pairs and that any nonadjacent vertices have the same number of common neighbors as other nonadjacent pairs.

Independence

In graph theory, the word independent usually carries the connotation of pairwise disjoint or mutually nonadjacent.
In this sense, independence is a form of immediate nonadjacency. An isolated vertex is a vertex not incident to any edges. An independent set, or coclique, or stable set or staset, is a set of vertices of which no pair is adjacent. Since the graph induced by any independent set is an empty graph, the two terms are usually used interchangeably. In the example above, vertices 1, 3, and 6 form an independent set; and 3, 5, and 6 form another one.
Two subgraphs are edge disjoint if they have no edges in common. Similarly, two subgraphs are vertex disjoint if they have no vertices (and thus, also no edges) in common. Unless specified otherwise, a set of disjoint subgraphs are assumed to be pairwise vertex disjoint.
The independence number α(G) of a graph G is the size of the largest independent set of G.
A graph can be decomposed into independent sets in the sense that the entire vertex set of the graph can be partitioned into pairwise disjoint independent subsets. Such independent subsets are called partite sets, or simply parts.
A graph that can be decomposed into two partite sets but not fewer is bipartite; three sets but not fewer, tripartite; k sets but not fewer, k-partite; and an unknown number of sets, multipartite. An 1-partite graph is the same as an independent set, or an empty graph. A 2-partite graph is the same as a bipartite graph. A graph that can be decomposed into k partite sets is also said to be k-colourable.
A complete multipartite graph is a graph in which vertices are adjacent if and only if they belong to different partite
sets. A complete bipartite graph is also referred to as a biclique; if its partite sets contain n and m vertices, respectively, then the graph is denoted K n,m.
A k-partite graph is semiregular if each of its partite sets has a uniform degree; equipartite if each partite set has the same size; and balanced k-partite if each partite set differs in size by at most 1 with any other.

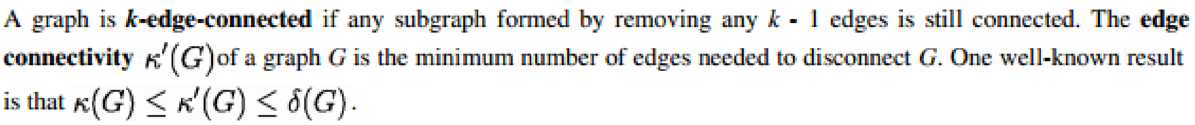

The **matching number** $\alpha'(G)$ of a graph G is the size of a largest **matching**, or pairwise vertex disjoint edges, of G.

A *spanning matching*, also called a **perfect matching** is a matching that covers all vertices of a graph.

Connectivity

Connectivity extends the concept of adjacency and is essentially a form (and measure) of concatenated adjacency.
If it is possible to establish a path from any vertex to any other vertex of a graph, the graph is said to be connected; otherwise, the graph is disconnected. A graph is totally disconnected if there is no path connecting any pair of vertices. This is just another name to describe an empty graph or independent set.

A cut vertex, or articulation point, is a vertex whose removal disconnects the remaining subgraph. A cut set, or vertex cut or separating set, is a set of vertices whose removal disconnects the remaining subgraph. A bridge is an analogous edge (see below).
If it is always possible to establish a path from any vertex to every other even after removing any k - 1 vertices, then the graph is said to be k-vertex-connected or k-connected. Note that a graph is k-connected if and only if it contains k internally disjoint paths between any two vertices. The example graph above is connected (and therefore 1-connected), but not 2-connected. The vertex connectivity or connectivity κ(G) of a graph G is the minimum number of vertices that need to be removed to disconnect G. The complete graph K n has connectivity n - 1 for n > 1; and a disconnected graph has connectivity 0.
In network theory, a giant component is a connected subgraph that contains a majority of the entire graph's nodes.
A bridge, or cut edge or isthmus, is an edge whose removal disconnects a graph. (For example, all the edges in a tree are bridges.) A disconnecting set is a set of edges whose removal increases the number of components. An edge cut is the set of all edges which have one vertex in some proper vertex subset S and the other vertex in V(G)\S. Edges of K 3 form a disconnecting set but not an edge cut. Any two edges of K 3 form a minimal disconnecting set as well as an edge cut. An edge cut is necessarily a disconnecting set; and a minimal disconnecting set of a nonempty graph is necessarily an edge cut. A bond is a minimal (but not necessarily minimum), nonempty set of edges whose removal disconnects a graph. A cut vertex is an analogous vertex (see above).



A graph is ***k*-edge-connected** if any subgraph formed by removing any k - 1 edges is still connected. The **edge connectivity** $\kappa'(G)$ of a graph G is the minimum number of edges needed to disconnect G. One well-known result is that $\kappa(G) \leq \kappa'(G) \leq \delta(G)$.

A component is a maximally connected subgraph. A block is either a maximally 2-connected subgraph, a bridge (together with its vertices), or an isolated vertex. A biconnected component is a 2-connected component.
An articulation point (also known as a separating vertex) of a graph is a vertex whose removal from the graph increases its number of connected components. A biconnected component can be defined as a subgraph induced by a maximal set of nodes that has no separating vertex.

Distance
The distance d G(u, v) between two (not necessary distinct) vertices u and v in a graph G is the length of a shortest path between them. The subscript G is usually dropped when there is no danger of confusion. When u and v are identical, their distance is 0. When u and v are unreachable from each other, their distance is defined to be infinity ∞.
The eccentricity ε G (v) of a vertex v in a graph G is the maximum distance from v to any other vertex. The diameter diam(G) of a graph G is the maximum eccentricity over all vertices in a graph; and the radius rad(G), the minimum.
When there are two components in G, diam(G) and rad(G) defined to be infinity ∞. Trivially, diam(G) ≤ 2 rad(G).
Vertices with maximum eccentricity are called peripheral vertices. Vertices of minimum eccentricity form the center. A tree has at most two center vertices.

The Wiener index of a vertex v in a graph G, denoted by WG(v) is the sum of distances between v and all others.
The Wiener index of a graph G, denoted by W(G), is the sum of distances over all pairs of vertices. An undirected graph's Wiener polynomial is defined to be Σ q d(u,v) over all unordered pairs of vertices u and v. Wiener index and Wiener polynomial are of particular interest to mathematical chemists.
The k-th power G k of a graph G is a supergraph formed by adding an edge between all pairs of vertices of G with distance at most k. A second power of a graph is also called a square.
A k-spanner is a spanning subgraph in which every two vertices are at most k times as far apart on S than on G. The number k is the dilation. k-spanner is used for studying geometric network optimization.

Genus

A crossing is a pair of intersecting edges. A graph is embeddable on a surface if its vertices and edges can be arranged on it without any crossing. The genus of a graph is the lowest genus of any surface on which the graph can embed.
A planar graph is one which can be drawn on the (Euclidean) plane without any crossing; and a plane graph, one which is drawn in such fashion. In other words, a planar graph is a graph of genus 0. The example graph is planar; the complete graph on n vertices, for n> 4, is not planar. Also, a tree is necessarily a planar graph.
When a graph is drawn without any crossing, any cycle that surrounds a region without any edges reaching from the cycle into the region forms a face. Two faces on a plane graph are adjacent if they share a common edge. A dual, or planar dual when the context needs to be clarified, G* of a plane graph G is a graph whose vertices represent the faces, including any outerface, of G and are adjacent in G * if and only if their corresponding faces are adjacent in G.
The dual of a planar graph is always a planar pseudograph (e.g. consider the dual of a triangle). In the familiar case of a 3-connected simple planar graph G (isomorphic to a convex polyhedron P), the dual G* is also a 3-connected simple planar graph (and isomorphic to the dual polyhedron P*).
Furthermore, since we can establish a sense of "inside" and "outside" on a plane, we can identify an "outermost" region that contains the entire graph if the graph does not cover the entire plane. Such outermost region is called an outer face. An outerplanar graph is one which can be drawn in the planar fashion such that its vertices are all adjacent to the outer face; and an outerplane graph, one which is drawn in such fashion.
The minimum number of crossings that must appear when a graph is drawn on a plane is called the crossing number.
The minimum number of planar graphs needed to cover a graph is the thickness of the graph.

Weighted graphs and networks

A weighted graph associates a label (weight) with every edge in the graph. Weights are usually real numbers. They may be restricted to rational numbers or integers. Certain algorithms require further restrictions on weights; for instance, Dijkstra's algorithm works properly only for positive weights. The weight of a path or the weight of a tree in a weighted graph is the sum of the weights of the selected edges. Sometimes a non-edge is labeled by a special weight representing infinity. Sometimes the word cost is used instead of weight. When stated without any qualification, a graph is always assumed to be unweighted. In some writing on graph theory the term network is a synonym for a weighted graph. A network may be directed or undirected, it may contain special vertices (nodes), such as source or sink. The classical network problems include:

- minimum cost spanning tree,
- shortest paths,
- maximal flow (and the max-flow min-cut theorem)

Direction

A directed arc, or directed edge, is an ordered pair of endvertices that can be represented graphically as an arrow drawn between the endvertices. In such an ordered pair the first vertex is called the initial vertex or tail; the second one is called the terminal vertex or head (because it appears at the arrow head). An undirected edge disregards any sense of direction and treats both endvertices interchangeably. A loop in a digraph, however, keeps a sense of direction and treats both head and tail identically. A set of arcs are multiple, or parallel, if they share the same head and the same tail. A pair of arcs are anti-parallel if one's head/tail is the other's tail/head. A digraph, or directed graph, or oriented graph, is analogous to an undirected graph except that it contains only arcs. A mixed graph may contain both directed and undirected edges; it generalizes both directed and undirected graphs. When stated without any qualification, a graph is almost always assumed to be undirected.

A digraph is called simple if it has no loops and at most one arc between any pair of vertices. When stated without ny qualification, a digraph is usually assumed to be simple.

In a digraph Γ, we distinguish the out degree d Γ + (v), the number of edges leaving a vertex v, and the in degree d Γ-(v), the number of edges entering a vertex v. If the graph is oriented, the degree d Γ (v) of a vertex v is equal to the sum of its out- and in- degrees. When the context is clear, the subscript Γ can be dropped. Maximum and minimum out degrees are denoted by Δ+ (Γ) and δ+ (Γ); and maximum and minimum in degrees, Δ- (Γ) and δ- (Γ). An out-neighborhood, or successor set, N + Γ (v) of a vertex v is the set of tails of arcs going from v. Likewise, an in-neighborhood, or predecessor set, N – Γ (v) of a vertex v is the set of heads of arcs going into v.

A source is a vertex with 0 in-degree; and a sink, 0 out-degree.

A vertex v dominates another vertex u if there is an arc from v to u. A vertex subset S is out-dominating if every vertex not in S is dominated by some vertex in S; and in-dominating if every vertex in S is dominated by some vertex not in S.

A kernel is an independent out-dominating set. A digraph is kernel perfect if every induced sub-digraph has a kernel.

An Eulerian digraph is a digraph with equal in- and out-degrees at every vertex.

The zweieck of an undirected edge is the pair of diedges and which form the simple dicircuit.

An orientation is an assignment of directions to the edges of an undirected or partially directed graph. When stated without any qualification, it is usually assumed that all undirected edges are replaced by a directed one in an orientation. Also, the underlying graph is usually assumed to be undirected and simple.

A tournament is a digraph in which each pair of vertices is connected by exactly one arc. In other words, it is an oriented complete graph.

A directed path, or just a path when the context is clear, is an oriented simple path such that all arcs go the same direction, meaning all internal vertices have in- and out-degrees 1. A vertex v is reachable from another vertex u if there is a directed path that starts from u and ends at v. Note that in general the condition that u is reachable from v does not imply that v is also reachable from u.

If v is reachable from u, then u is a predecessor of v and v is a successor of u. If there is an arc from u to v, then u is a direct predecessor of v, and v is a direct successor of u.
A digraph is strongly connected if every vertex is reachable from every other following the directions of the arcs.
On the contrary, a digraph is weakly connected if its underlying undirected graph is connected. A weakly connected graph can be thought of as a digraph in which every vertex is "reachable" from every other but not necessarily following the directions of the arcs. A strong orientation is an orientation that produces a strongly connected digraph.
A directed cycle, or just a cycle when the context is clear, is an oriented simple cycle such that all arcs go the same direction, meaning all vertices have in- and out-degrees 1. A digraph is acyclic if it does not contain any directed cycle. A finite, acyclic digraph with no isolated vertices necessarily contains at least one source and at least one sink.
An arborescence, or out-tree or branching, is an oriented tree in which all vertices are reachable from a single vertex. Likewise, an in-tree is an oriented tree in which a single vertex is reachable from every other one.

Directed acyclic graphs

The partial order structure of directed acyclic graphs (or DAGs) gives them their own terminology. If there is a directed edge from u to v, then we say u is a parent of v and v is a child of u. If there is a directed path from u to v, we say u is an ancestor of v and v is a descendant of u.
The moral graph of a DAG is the undirected graph created by adding an (undirected) edge between all parents of the same node (sometimes called marrying), and then replacing all directed edges by undirected edges. A DAG is perfect if, for each node, the set of parents is complete (i.e. no new edges need to be added when forming the moral graph).

Colouring

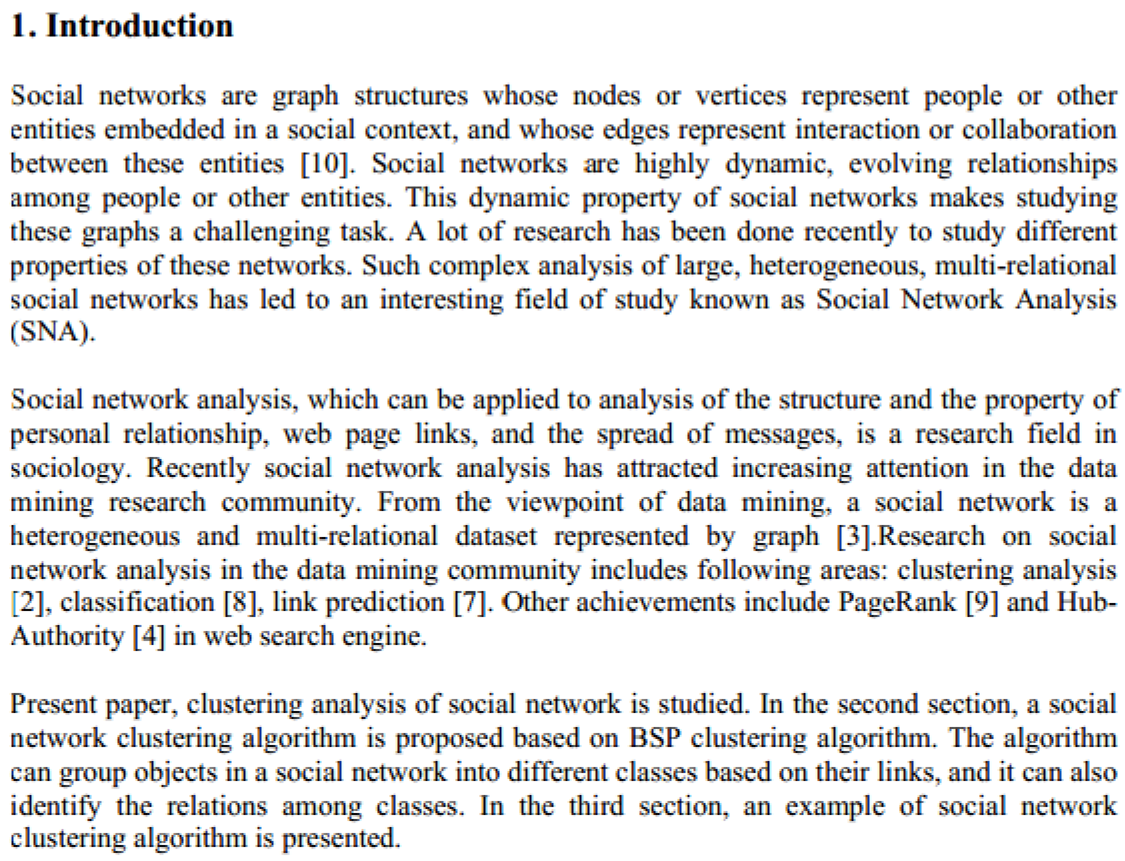
Vertices in graphs can be given colours to identify or label them. Although they may actually be rendered in diagrams in different colours, working mathematicians generally pencil in numbers or letters (usually numbers) to represent the colours.
Given a graph G (V,E) a k-colouring of G is a map $\phi : V \rightarrow \{1, ... , k\}$ with the property that $(u, v) \in E \Rightarrow \phi(u) \neq \phi(v)$ - in other words, every vertex is assigned a colour with the condition that adjacent vertices cannot be assigned the same colour.
The chromatic number $\chi(G)$ is the smallest k for which G has a k-colouring.
Given a graph and a colouring, the colour classes of the graph are the sets of vertices given the same colour.

Various

A graph invariant is a property of a graph G, usually a number or a polynomial, that depends only on the isomorphism class of G. Examples are the order, genus, chromatic number, and chromatic polynomial of a graph.

Improved BSP Clustering Algorithm for Social Network Analysis

Social network analysis is a new research field in data mining. The clustering in social network analysis is different from traditional clustering. It requires grouping objects into classes based on their links as well as their attributes. While traditional clustering algorithms group objects only based on objects' similarity, and it can't be applied to social network analysis. So on the basis of BSP (business system planning) clustering algorithm, a social network clustering analysis algorithm is proposed. The proposed algorithm, different from traditional BSP clustering algorithms, can group objects in a social network into different classes based on their links and identify relation among classes dynamically & require less amount of memory .



1. Introduction

Social networks are graph structures whose nodes or vertices represent people or other entities embedded in a social context, and whose edges represent interaction or collaboration between these entities [10]. Social networks are highly dynamic, evolving relationships among people or other entities. This dynamic property of social networks makes studying these graphs a challenging task. A lot of research has been done recently to study different properties of these networks. Such complex analysis of large, heterogeneous, multi-relational social networks has led to an interesting field of study known as Social Network Analysis (SNA).

Social network analysis, which can be applied to analysis of the structure and the property of personal relationship, web page links, and the spread of messages, is a research field in sociology. Recently social network analysis has attracted increasing attention in the data mining research community. From the viewpoint of data mining, a social network is a heterogeneous and multi-relational dataset represented by graph [3].Research on social network analysis in the data mining community includes following areas: clustering analysis [2], classification [8], link prediction [7]. Other achievements include PageRank [9] and Hub-Authority [4] in web search engine.

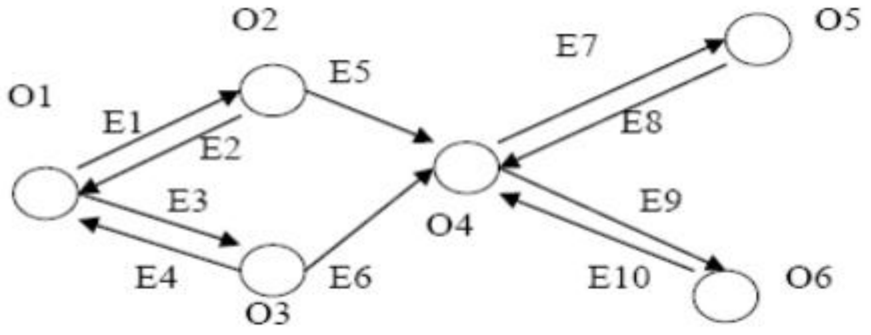
Present paper, clustering analysis of social network is studied. In the second section, a social network clustering algorithm is proposed based on BSP clustering algorithm. The algorithm can group objects in a social network into different classes based on their links, and it can also identify the relations among classes. In the third section, an example of social network clustering algorithm is presented.

2. Social Network Analysis Based on BSP Clustering

There has been extensive research work on clustering in data mining. Traditional clustering algorithms [3] divide objects into classes based on their similarity. Objects in a class are similar to each other and are very dissimilar from objects in different classes. Social network clustering analysis, which is different from traditional clustering problem, divides objects into classes based on their links as well as their attributes. The biggest challenge of social network clustering analysis is how to divide objects into classes based on objects' links, thus find algorithms that can meet this challenge.

The BSP (business system planning) clustering algorithm [11] is proposed by IBM. It designed to define information architecture for the firm in business system planning. This algorithm analyses business process and their data classes, cluster business process into sub-systems, and define the relationship of these sub-systems.

Basically BSP clustering algorithm uses objects (business processes) and links among objects (data classes) to make clustering analysis. Similarly social network also includes objects and links among these objects. In view of the same pre-condition, the BSP clustering algorithm can be used in social network clustering analysis.According to graph theory, social network is a directed graph composed by objects and their relationship. Figure 1 shows a sample of social network, the circle in the figure represents an object; the line with arrow is an edge of the graph, and it represents directed link between two objects, so a social network is a directed graph.



In figure 1, Let *Oi* be an object in social network (*i* = 1...*m*), let E_j which means directed link between two objects, be a directed edge of the graph (*j* = 1...*n*).

After definition of objects and directed edges, also define reachable relation between two objects. There are two kinds of reachable relation among objects, shown as following:

1) One-step reachable relation: if there has directed link from *Oi* to O_j through one and only one directed edge, then O*i* to *Oj* is a one-step reachable relation. For instance in figure 1 there has a directed link from O_1 to O_2 through the directed edge E_1, O_1 to O_2 is one-step reachable relation.

2) Multi-steps reachable relation: if there has directed link from O_i to O_j through two or more directed edges, then *O* i to O_j is a multi-steps reachable relation. For instance in figure 1

has a directed link from O_1 to O_4 through directed edges E_1 and E_5, then O1 to O4 is a 2-steps reachable relation.
After these definitions, use BSP clustering algorithm to analyses a social network. The analysis processes are as following steps:

Generate edge creation matrix and edge pointed matrix
First according to the objects and edges in the graph, define two matrixes *Lc* and *Lp*.
Let *Lc* be a $m\times n$ matrix which means the creation of edges. In the matrix, $Lc\ (i, j)$ =1 denotes object *Oi* connects with the tail of edge *Ej* , which means that object *Oi* creates the directed edge *Ej* . $L\ (i, j)\ c$ =0 denotes *Oi* doesn't connect with the tail of edge *Ej* , which means *Ej* isn't created by object *Oi* .

Calculate one-step reachable matrix between objects

After the definition of *Lc* and *Lp*, calculate one-step reachable matrix between objects through the following equation.

$$G=Lc\ *L_p^T = g_{i,j}=\bigvee_{k=1}^{n}(l_c(i,k)\wedge l_p^T(k,j))$$

$$,i=1.....m,\left[j=1......m\right] \tag{1}$$

^ is Boolean product, V is Boolean sum.
$G(i, j)$ =1 means *Oi* to *Oj* is a one-step reachable relation, $G(i, j)$ = 0 means there hasn't a one-step reachable relation from *Oi* to *Oj* . Through G, calculate all one-step reachable relation between objects.

Calculate multi-steps reachable matrix between objects

Besides one-step reachable relation, there are multi-steps reachable relations between objects too. Also need calculate multi-steps reachable matrices (2-steps, 3-steps, …,*m*-1-steps).
According to graph theory and the BSP clustering algorithm, calculate multi-steps reachable matrix *G*2, G3, G4,…., *Gm*−1 . Following equations show the calculation of multi-steps reachable matrix:

$$G^2=G\ *G = \quad g^2_{i,j}=\bigvee_{k=1}^{m}(g(i,k)\wedge g(k,j))$$

$$,i=1.....m,\left[j=1......m\right] \tag{2}$$

$$G^3 = G^2 * G$$
$$G^4 = G^3 * G$$
……
$$G^{m-1} = G^{m-2} * G$$

These matrices include 2-steps, 3-steps… *m*-1-steps reachable relations between objects. Now *n*-steps reachable relation between two objects through *G*2 ,*G*3 ,*G*4 ,...,*Gm*−1 .

Calculate reachable matrix

Because only consider whether reachable relations exist between two objects, but do not care these relations are one-step or multi-steps, so calculate reachable matrix R based on $G,G2$,$G3$,$G4$,...,$Gm-1$. The calculation of R is shown as following equation:

$$R=IVGVG2...VGm-1 \qquad (3)$$

V is Boolean sum, I is unit matrix.
$R(i, j) = 1$ means reachable relation exists from Oi to Oj , but the reachable relations existing in matrix R is not mutual, for instance $R(i, j) = 1$ means reachable relation exists from Oi to Oj ,but it doesn't means reachable relation exists from Oj to Oi . Mutual reachable relations between two objects are important in a social network, so
calculate mutual reachable matrix based on R .

Calculate mutual reachable matrix and generate clusters
The mutual reachable matrix can be calculated through following calculate equation.

$$Q=R\hat{}RT \qquad (4)$$

^means Boolean product

In the matrix$Q(i, j) = 1$ means there are mutual reachable relation between Oi and Oj . In a social network if two objects that have mutual reachable relation, they should belong to the same class, thus cluster based on Q.
Thus according to mutual reachable matrix Q, divide a social network into classes based on strong submatrices in Q or adjusted Q. While strong sub-matrix is defined as follows.

Strong sub-matrix: if all elements in a sub-matrix of Q are 1, this sub matrix is strong sub-matrix.

Identify relationships among classes
After clustering of social network, also need identify relationship among clusters. This can be done through generated clusters and one-step reachable matrix$G$. If there is one-step reachable relation between two objects in different classes, so directed links exist between classes. Through$G$, identify all relations among classes.
After pervious 6 steps, divide a social network into classes. Social network clustering analysis algorithm can be given:

```
Input:
 $L_c$ : Edge creation Matrix
$L_p$ : Edge pointed matrix
Begin
 $G = L_c * L_u{}^T$
for k=3 to m do
$Gk -1 =Gk -2 *G$
$R = I V G V G2 ... V Gm-1$
$Q = R \hat{} R^T$
 $Q_k - > C$
$(C_k ,Q)$->Relation $(C_k)$
End
```

$Q- > C_k$ means generating clusters through mutual reachable matrix$Q$, and (C_k ,Q)->Relation(C_k) means identifying relationships among clusters base on clusters and one-step reachable matrix$G$.

3. Improvement over BSP Clustering Algorithm

In previous paper based on BSP clustering algorithm, an algorithm of social network clustering analysis is proposed. It divides a social network into different classes according to objects in the social network and links between objects, and it also can identify relations among clusters.

Main disadvantage of this algorithm is that it uses matrices to store edges and reachable relations, in a real social network these matrices will be very huge, can't load into main memory. But because these matrices are very sparse, so design an efficient data structure to overcome this shortcoming.

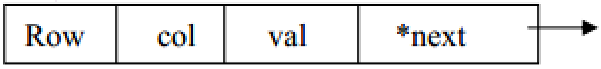
Present paper propose modification of existing BSP algorithm using Link list data structure. Using this data structure can overcome shortcoming (which have been mention above) .Following procedure is require for converting this sparse metrix in to link list:

A matrix is a two-dimensional data object made of m rows and n columns, therefore having *m*, *n* values. When *m*=*n*, call it a square matrix.

The most natural representation is to use two-dimensional array A[m][n] and access the element of ith row and jth column as A[i][j]. If a large number of elements of the matrix are zero elements, then it is called a sparse matrix.

Representing a sparse matrix by using a two-dimensional array leads to the wastage of a substantial amount of space. Therefore, an alternative representation must be used for sparse matrices. One such representation is to store only non- zero elements along with their row positions and column positions. That means representing every non-zero element by using triples (i, j, value), where i is a row position and j is a column position, and store these triples in a linear list. It is possible to arrange these triples in the increasing order of row indices, and for the same row index in the increasing order of column indices. Each triple (i,j,value) can be represented by using a node having four fields as shown in the following:

```
Struct snode{
     Int row,col,val;
    Struct snode *next;
    };
```



1. In order to add two sparse matrices represented using the sorted linked lists as shown in the preceding program, the lists are traversed until the end of one of the lists is reached.
2. In the process of traversal, the row indices stored in the nodes of these lists are compared. If they don't match, a new node is created and inserted into the resultant list by copying the contents of a node with a lower value of row index. The pointer in the list containing a node with a lower value of row index is advanced to make it point to the next node.
3. If the row indices match, column indices for the corresponding row positions are compared. If they don't match, a new node is created and inserted into the resultant list by copying the contents of a node with a lower value of column index. The pointer in the list containing a node with a lower value of column index is advanced to make it point to the next node.

4. If the column indices match, a new node is created and inserted into the resultant list by copying the row and column indices from any of the nodes and the value equal to the sum of the values in the two nodes.
5. After this, the pointers in both the lists are advanced to make them point to the next nodes in the respective lists. This process is repeated in each iteration. After reaching the end of any one of the lists, the iterations come to an end and the remaining nodes in the list whose end has not been reached are copied, as it is in the resultant list.

After pervious 5 steps, divide a social network into classes. Social network clustering analysis algorithm can be given:

```
Input:
 Lu : Edge creation Lists
 Lp : Edge pointed Lists
Begin
$G = L_c$ * Swap ($L_u$ ) //perform swapping row column of $L_u$
for  k=3 to m do
$Gk -1 = Gk -2 * G$
Lp : Edge pointed Lists
Begin
$G = L_c$ * Swap ($L_u$ ) //perform swapping row column of $L_u$
for  k=3 to m do
$Gk -1 = Gk -2 * G$
$R = I \vee G \vee G2 \ldots \vee Gm-1$
$Q = R \wedge Swap(R)$ //perform swapping row column of R
```

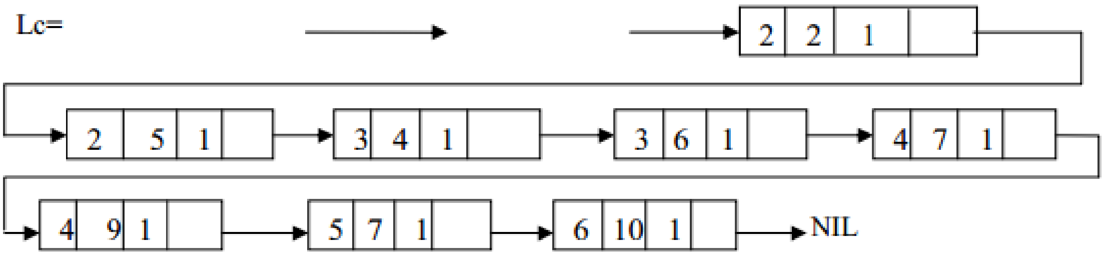
Working Example

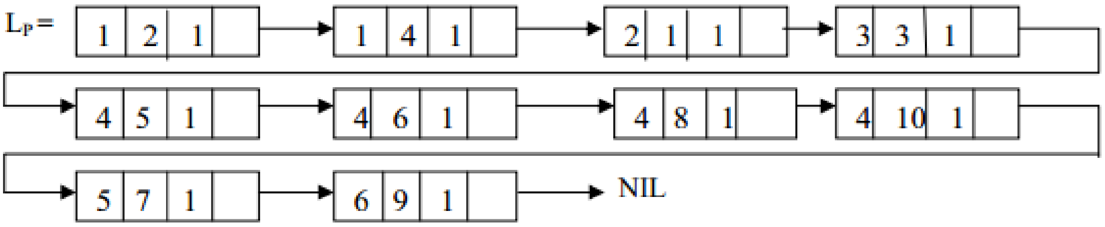
Now an example is given to show process of the cluster analysis of social network. Suppose a social network as figure 1 shows. According to the figure, can give the edge creation matrix *Lc* and edge po inted matrix *Lp* as following.

In the form of link list

1	1	1	

1	3	1	

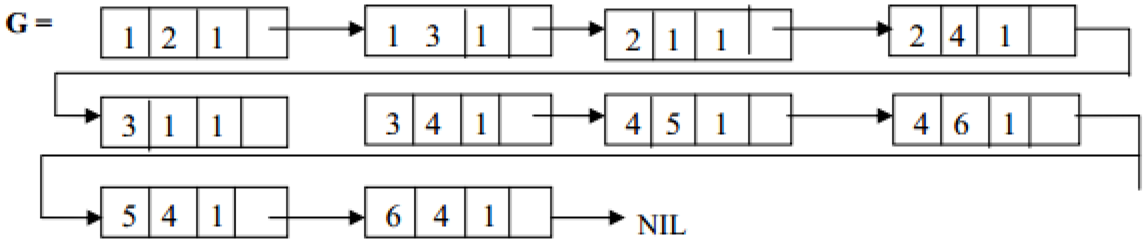




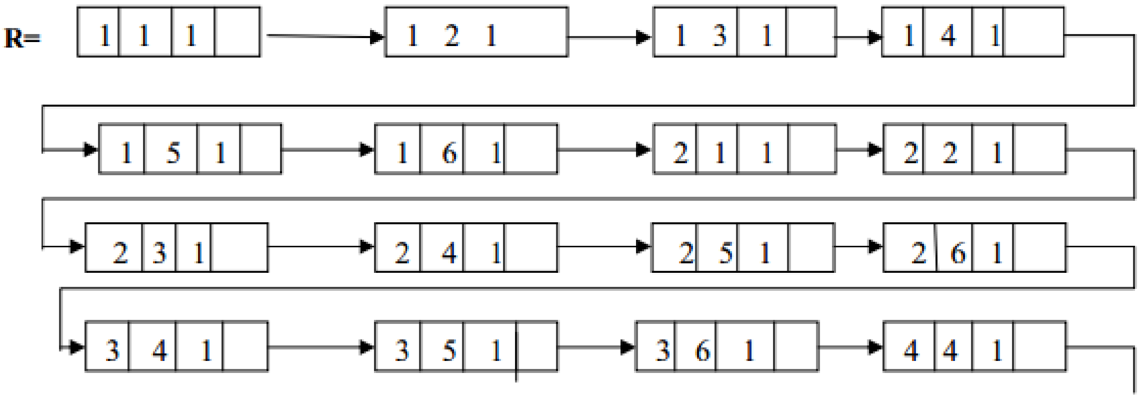
According to the social network clustering algorithm, *Lc* and *Lp* clustering the social network show as following steps:

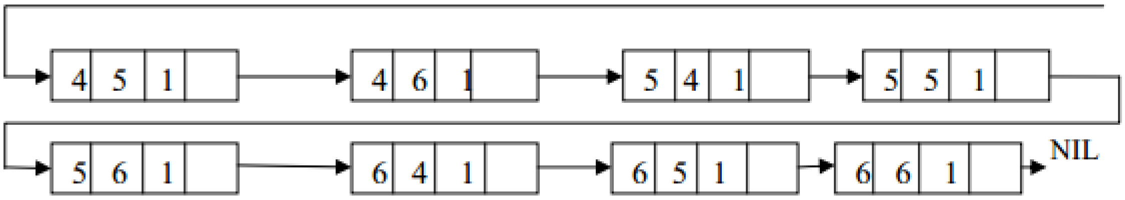
Calculate one-step reachable matrix between objects

$$G = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \bullet \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}^T = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$



Calculate reachable matrix based on one-step and multi-steps reachable matrix

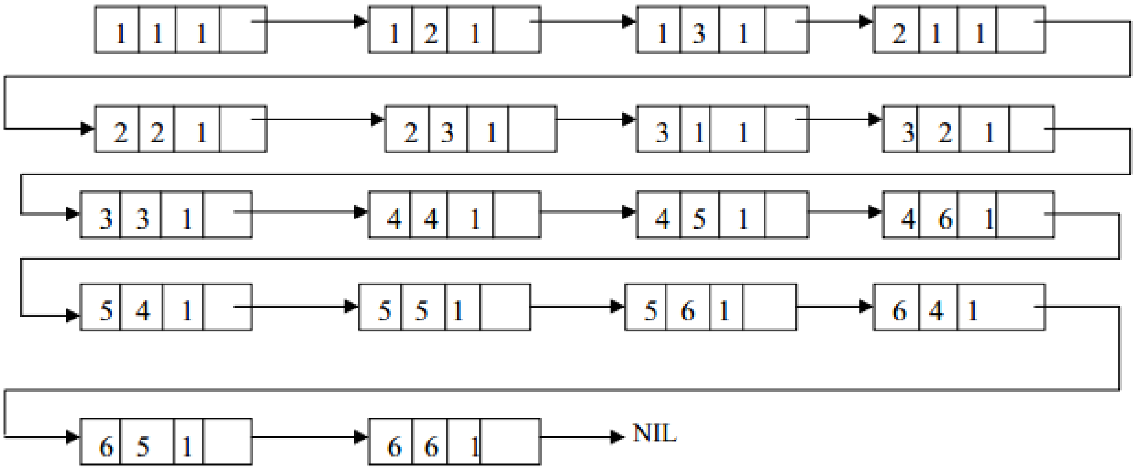




Calculate mutual reachable matrix and generate clusters

$$Q = R \wedge R^{T} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \wedge \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

Q=



According the mutual reachable list Q, it includes two strong sub list. So divide figure 1 to two classes, the first class C_1 includes object O_1 ,O_2 ,O_3 , and the second class C_2 includes O_4 ,O_5 ,O_6 .

Identifying relationships among classes

According to one-step reachable matrix G , there have one-step reachable relations between two classes ($O_2 -> O_4$ and $O_3 -> O_4$), so identify relations between two clusters C_1 and C_2 , as figure 2 shows.

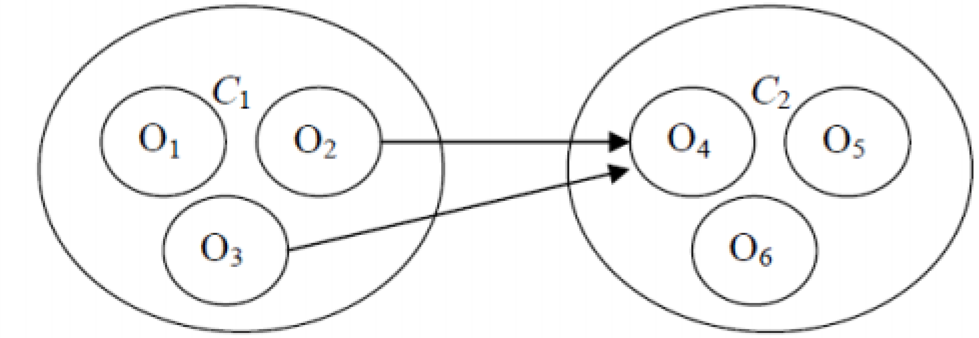


Figure 2: Identify relationships between two clusters.

C points to C_2 *in* figure 2, but C_2 not points to C_1, so identify relations between two classes.

4. Conclusion

In this paper based on BSP clustering algorithm, an algorithm of social network clustering analysis is proposed. It divides a social network into different classes according to objects in the social network and links between objects, and it also can identify relations among clusters. Also in our algorithm the edges between objects have same weight; however in real world such edges may have different weights. Meanwhile the property of each cluster has not been analyzed. These will be solved in our future research.

PROGRAM CODE:

```
#include<stdlib.h>
#include<malloc.h>
#include<stdio.h>
#include<conio.h>
#include<dos.h>
#include<time.h>
clock_t start, end;
int count5[5];
float tim,tim2;
int b[4];
int x2,y;
int recieve(int[],int,char);
void lin2()
{       int i;
        for(i=18;i<50;i++)
        {       gotoxy(40,i);
                printf("%c\n",186);

        }
}
void lin11()
{
        int i;
        for(i=4;i<17;i++)
        {
                gotoxy(50,i);
                printf("%c\n",186);
        }
}

void title(void)
{
        int i;

        gotoxy(2,1);

        for(i=0;i<78;i++)
        {
                printf("%c",205);
        }
        gotoxy(30,2);
        printf("%c",4);
        printf("%c",4);
        printf("STOP 'N' WAIT PROTOCOL");
        printf("%c",4);
        printf("%c",4);

        gotoxy(2,3);
        for(i=0;i<78;i++)
        {
             printf("%c",205);
        }

   }
   void menu1()
   {
        int i,j;
        title();
        printf("\n\n");
        printf("\n\n\t1. Normal Sending Data");
        printf("\n\t2. Loss of Acknowledgement\
             \n\t3. Loss of Sending Data\
             \n\t4. Time Expired\
             \n\t5. Exit");
        printf("\n\n\tEnter your choice : ");
        gotoxy(55,4);
        printf("\tMASTER TABLE");
        for(i=6;i<17;i++)
        {
             gotoxy(65,i);
             printf("%c\n",186);
        }
        gotoxy(51,5);
        for(i=0;i<29;i++)
        {
             printf("%c",205);
        }
        j=1;
        for(i=7;i<=15;i++)
        {
             gotoxy(55,i);
             printf("Frame %d",j);
             j++;
             i++;
        }
        gotoxy(2,17);
        for(i=0;i<78;i++)
        {
             printf("%c",205);
        }
   }
   void menu(void)
   {

        int i;
        gotoxy(2,17);
        printf("\n\n");
        printf("\t      SENDER'S END ",127,127);
        printf("\t\t\t  RECIEVER'S END \n\n");
        lin2();
        lin11();
   }
   void send(char z)
   {
        int a[20],i,x1,x,count=1;
        int f[4];
```

```
time_t t;
float tim1=0.0;
//char *a1;
//int l=0,*b1;
int j=0,k=21,cnt=7,ct=7,cnt1=0,g;
for(i=0;i<4;i++)
{
      b[i]=2;
      count5[i]=100;
}
count5[i]=100;
gotoxy(4,21);
printf("Message : ");
srand((unsigned) time(&t));
x2=14;
y=21;
for(i=0;i<20;i++)
{
      gotoxy(x2,y);
      a[i]=(rand() %2);
      printf("%d",a[i]);
      x2++;
}
ct=7;
for(g=0;g<=4;g++)
{
      gotoxy(69,ct);
      ct=ct+2;
      printf(" %d",count5[g]);
}
gotoxy(4,23);
printf("Press Enter To Send Data");
getch();
switch(z)
{
      case '1':

            for(i=0; i<20; i++)
            {
                  if(i!=0)
                  {
                        clrscr();
                        menu1();
                        menu();
                        gotoxy(4,21);
                        printf("Message : ");
                        x2=14;
                        y=21;
                        for(j=0;j<20;j++)
                        {
                              gotoxy(x2,y);
                              printf("%d",a[j]);
                              x2++;
                        }
                        ct=7;
                        for(g=0;g<=4;g++)
                  {
                        gotoxy(69,ct);
                        ct=ct+2;
                        printf(" %d",count5[g]);
                  }
            }
            x2=4;
            y=23;
            for(k=0;k<4;k++)
            {
                  f[k]=a[i];
                  i++;
            }
            i--;
            y=y+1;
            gotoxy(x2,y);
            printf("\tSending Frame %d : ",count);
            for(k=0;k<4;k++)
                  printf("%d",f[k]);
            count5[cnt1]--;
            ct=7;
            for(g=0;g<=cnt1;g++)
            {
                  gotoxy(69,ct);
                  ct=ct+2;
                  printf("          ");
            }
            ct=7;
            for(g=0;g<=cnt1;g++)
            {
                  gotoxy(69,ct);
                  ct=ct+2;
                  printf(" %d",count5[g]);
            }
            cnt++;
            cnt++;
            cnt1++;
            start = clock();
            x=recieve(f,count,z);
            x2=x2-40;
            y+=2;
            count++;
            if(i<16)
            {
                  gotoxy(x2,y);
                  printf("\tPress Enter to Continue");
                  getch();
            }
      }
      y+=1;
      gotoxy(x2,y);
      printf("\tMessage ");
      printf(" Sent Successfully");
      break;
case '2':
      for(i=0; i<20; i++)
```

```
        {
            for(k=0;k<4;k++)
                b[k]=2;
            if(i!=0)
            {
                clrscr();
                menu1();
                menu();
                gotoxy(4,21);
                printf("Message : ");
                x2=14;
                y=21;
                for(j=0;j<20;j++)
                {
                    gotoxy(x2,y);
                    printf("%d",a[j]);
                    x2++;
                }
                ct=7;
                for(g=0;g<=4;g++)
                {
                    gotoxy(69,ct);
                    ct=ct+2;
                    printf(" %d",count5[g]);
                }
            }
            x2=4;
            y=23;
            for(k=0;k<4;k++)
            {
                f[k]=a[i];
                i++;
            }
            i--;
            gotoxy(x2,y);
            printf("\tSending Frame %d : ",count);
            for(k=0;k<4;k++)
                printf("%d",f[k]);
            count5[cnt1]--;
            ct=7;
            for(g=0;g<=cnt1;g++)
            {
                gotoxy(69,ct);
                ct=ct+2;
                printf("          ");
            }

            ct=7;
            for(g=0;g<=cnt1;g++)
            {
                gotoxy(69,ct);
                ct=ct+2;
                printf(" %d",count5[g]);
            }
            delay(100);
            x=recieve(f,count,z);
```

```
                      x2=x2-40;
                      y+=2;
                      tim1=0.0;
                      while(x==1)
                      {
                            end=clock();
                            tim=(end-start)/CLK_TCK;
                            tim1=tim1+tim;
                            if(tim>1.3)
                            {
                                  gotoxy(x2,y);
                                  printf("\tAcknowledgement Not
Recieved");
                                  y=y+1;
                                  gotoxy(x2,y);
                                  printf("\tTotal Taken : %.2f
sec",tim1);
                                  y=y+1;
                                  gotoxy(x2,y);
                                  printf("\tResending Frame %d :
",count);
                                  for(k=0;k<4;k++)
                                        printf("%d",f[k]);
                                  ct=7;
                                  count5[cnt1]--;
                                  for(g=0;g<=cnt1;g++)
                                  {
                                        gotoxy(69,ct);
                                        ct=ct+2;
                                        printf("          ");
                                  }
                                  ct=7;
                                  for(g=0;g<=cnt1;g++)
                                  {
                                        gotoxy(69,ct);
                                        ct=ct+2;
                                        printf(" %d",count5[g]);
                                  }
                                  delay(100);
                                  x=recieve(f,count,z);
                                  x2=x2-40;
                                  y+=2;
                            }
                            else
                            {
                                  gotoxy(x2,y);
                                  printf("Acknowledgement Recieved");
                                  y=y+1;
                                  gotoxy(x2,y);
                                  printf("\tTotal Time  : %.2f
sec",tim1);
                                  break;
                            }
                      }
                      cnt++;
                      cnt++;
```

```
                    cnt1++;
                    count++;
                    if(i<16)
                    {
                          gotoxy(x2,y+2);
                          printf("\tPress Enter to Continue");
                          getch();
                    }
              }
              y+=2;
              gotoxy(x2,y);
              printf("\tMessage Sent Successfully");
              break;
        case '3':
              for(i=0; i<20; i++)
              {
                    tim2=0.0;
                    for(k=0;k<4;k++)
                          b[k]=2;
                    if(i!=0)
                    {
                          clrscr();
                          menu1();
                          menu();
                          gotoxy(4,21);
                          printf("Message : ");
                          x2=14;
                          y=21;
                          for(j=0;j<20;j++)
                          {
                                gotoxy(x2,y);
                                printf("%d",a[j]);
                                x2++;
                          }
                          ct=7;
                          for(g=0;g<=4;g++)
                          {
                                gotoxy(69,ct);
                                ct=ct+2;
                                printf(" %d",count5[g]);
                          }
                    }
                    x2=4;
                    y=23;
                    for(k=0;k<4;k++)
                    {
                          f[k]=a[i];
                          i++;
                    }
                    i--;
                    gotoxy(x2,y);
                    printf("\tSending Frame %d : ",count);
                    for(k=0;k<4;k++)
                          printf("%d",f[k]);
                    count5[cnt1]--;
                    ct=7;
```

```
for(g=0;g<=cnt1;g++)
{
      gotoxy(69,ct);
      ct=ct+2;
      printf("          ");
}
ct=7;
for(g=0;g<=cnt1;g++)
{
      gotoxy(69,ct);
      ct=ct+2;
      printf(" %d",count5[g]);
}
start = clock();
delay(100);
x=recieve(f,count,z);
x2=x2-40;
y+=2;
while(x==0)
{
      gotoxy(x2,y);
      printf("\tFrame Lost");
      y=y+1;
      gotoxy(x2,y);
      printf("\tResending Frame %d : ",count);
      for(k=0;k<4;k++)
            printf("%d",f[k]);
      count5[cnt1]--;
      ct=7;
      for(g=0;g<=cnt1;g++)
      {
            gotoxy(69,ct);
            ct=ct+2;
            printf("          ");
      }
      ct=7;
      for(g=0;g<=cnt1;g++)
      {
            gotoxy(69,ct);
            ct=ct+2;
            printf(" %d",count5[g]);
      }
      delay(100);
      start=clock();
      x=recieve(f,count,z);
      x2=x2-40;
      y+=2;
}
cnt++;
cnt++;
cnt1++;
count++;
if(i<16)
{
      gotoxy(x2,y+2);
      printf("\tPress Enter to Continue");
```

```
                  getch();
            }
      }
      y+=2;
      gotoxy(x2,y);
      printf("\tMessage Sent Successfully");
      break;

case '4':
      for(i=0; i<20; i++)
      {
            tim2=0.0;
            for(k=0;k<4;k++)
                  b[k]=2;
            if(i!=0)
            {
                  clrscr();
                  menu1();
                  menu();
                  gotoxy(4,21);
                  printf("Message : ");
                  x2=14;
                  y=21;
                  for(j=0;j<20;j++)
                  {
                        gotoxy(x2,y);
                        printf("%d",a[j]);
                        x2++;
                  }
                  ct=7;
                  for(g=0;g<=4;g++)
                  {
                        gotoxy(69,ct);
                        ct=ct+2;
                        printf(" %d",count5[g]);
                  }
            }
            x2=4;
            y=23;
            for(k=0;k<4;k++)
            {
                  f[k]=a[i];
                  i++;
            }
            i--;
            gotoxy(x2,y);
            printf("\tSending Frame %d : ",count);
            for(k=0;k<4;k++)
                  printf("%d",f[k]);
            count5[cnt1]--;
            ct=7;
            for(g=0;g<=cnt1;g++)
            {
                  gotoxy(69,ct);
                  ct=ct+2;
                  printf("          ");
```

```
        }
        ct=7;
        for(g=0;g<=cnt1;g++)
        {
                gotoxy(69,ct);
                ct=ct+2;
                printf(" %d",count5[g]);
        }
        start = clock();
        delay(100);
        x=recieve(f,count,z);
        x2=x2-40;
        y+=2;
        while(x==0)
        {
                gotoxy(x2,y);
                printf("\tTime Expired");
                y=y+1;
                gotoxy(x2,y);
                printf("\tResending Frame %d : ",count);
                for(k=0;k<4;k++)
                        printf("%d",f[k]);
                count5[cnt1]--;
                ct=7;
                for(g=0;g<=cnt1;g++)
                {
                        gotoxy(69,ct);
                        ct=ct+2;
                        printf("         ");
                }
                ct=7;
                for(g=0;g<=cnt1;g++)
                {
                        gotoxy(69,ct);
                        ct=ct+2;
                        printf(" %d",count5[g]);
                }
                delay(100);
                start=clock();
                x=recieve(f,count,z);
                x2=x2-40;
                y+=2;
        }
        cnt++;
        cnt++;
        cnt1++;
        count++;
        if(i<16)
        {
                gotoxy(x2,y+2);
                printf("\tPress Enter to Continue");
                getch();
        }
}
y+=2;
gotoxy(x2,y);
```

```
                  printf("\tMessage Sent Successfully");
                  break;

      }
}
int recieve(int f[],int i,char z)
{
      time_t t;
      int a,j;
      int c=0;
      //srand((unsigned) time(&t));
      //c=(rand()%2000);
      //delay(c);
      //end=clock();
      //
      switch(z)
      {
            case '1':
                  srand((unsigned) time(&t));
                  c=(rand()%1301);
                  delay(c);
                  end=clock();
                  tim=(end-start)/CLK_TCK;
                  x2=x2+40;
                  gotoxy(x2,y);
                  printf("\tFrame %d Recieved : ",i);
                  for(j=0;j<4;j++)
                        printf("%d",f[j]);
                  gotoxy(x2,y+1);
                        printf("\t Time : %.2f sec",tim);
                  return 1;
            case '2':
                  x2=x2+40;
                  for(j=0;j<4;j++)
                  {
                        if(b[j]==f[j])
                              c=1;
                        else
                        {
                              c=0;
                              break;
                        }
                  }
                  if(c==1)
                  {
                        gotoxy(x2,y);
                        printf("\tFrame Recieved before");
                        y=y+1;
                        gotoxy(x2,y);
                        printf("\tDscarding this Message");
                        y=y+1;
                        gotoxy(x2,y);
                        printf("\tSending Acknowledgement Again");
                  }
                  else
                  {
```

```
                gotoxy(x2,y);
                printf("\tFrame %d Recieved : ",i);
                for(j=0;j<4;j++)
                    printf("%d",f[j]);
                y=y+1;
                gotoxy(x2,y);
                printf("\tSending Acknowlegement");
                for(j=0;j<4;j++)
                    b[j]=f[j];
            }
            start=clock();
            srand((unsigned) time(&t));
            c=(rand()%2000);
            delay(c);
            return 1;
        case '3':
            x2=x2+40;
            srand((unsigned) time(&t));
            c=(rand()%2000);
            delay(c);
            end=clock();
            tim=(end-start)/CLK_TCK;
            tim2=tim2+tim;
            if(tim<1.3)
            {
                for(j=0;j<4;j++)
                    b[j]=f[j];
                gotoxy(x2,y);
                printf("\tFrame %d Recieved : ",i);
                for(j=0;j<4;j++)
                    printf("%d",f[j]);
                y=y+1;
                gotoxy(x2,y);
                printf("\tTotal Time : %.2f sec",tim2);
                return 1;
            }
            else
            {
                gotoxy(x2,y);
                printf("\tFrame %d Lost",i);
                y=y+1;
                gotoxy(x2,y);
                printf("\tTotal Time : %.2f sec",tim2);
                return 0;
            }
        case '4':
            x2=x2+40;
            srand((unsigned) time(&t));
            c=(rand()%2000);
            delay(c);
            end=clock();
            tim=(end-start)/CLK_TCK;
            tim2=tim2+tim;
            if(tim<1.3)
            {
                gotoxy(x2,y);
                printf("\tFrame %d Recieved : ",i);
                for(j=0;j<4;j++)
                    printf("%d",f[j]);
                y=y+1;
                gotoxy(x2,y);
                printf("\tTotal Time : %.2f sec",tim2);
                return 1;
            }
            else
            {
                y=y+1;
                gotoxy(x2,y);
                printf("\tTotal Time : %.2f sec",tim2);
                return 0;
            }
    }
}

void main()
{
    char ch;
    while(1)
    {
        clrscr();
        menu1();
        menu();
        gotoxy(28,13);
        fflush(stdin);
        ch=getche();
        switch(ch)
        {
            case '1':
                send(ch);
                getch();
                break;
            case '2':
                send(ch);
                getch();
                break;
            case '3':
                send(ch);
                getch();
                break;
            case '4':
                send(ch);
                getch();
                break;
            case '5':
                exit(0);
            default:
                printf("\n\n\t\t!!!!!Wrong Choice!!!!!");
                getch();
        }
    }
}
```

REFERENCES

[1] Linton Freeman, The Development of Social Network Analysis. Vancouver: Empirical Press, 2006.
[2] Wellman, Barry and S.D. Berkowitz, eds., 1988. Social Structures: A Network Approach. Cambridge: Cambridge University Press.
[3] Hansen, William B. and Reese, Eric L. 2009. Network Genie User Manual (https:/ / secure. networkgenie. com/ admin/ documentation/ Network_Genie_Manual. pdf). Greensboro, NC: Tanglewood Research.
[4] Freeman, Linton. 2006. The Development of Social Network Analysis. Vancouver: Empirical Pres, 2006; Wellman, Barry and S.D. Berkowitz, eds., 1988. Social Structures: A Network Approach. Cambridge: Cambridge University Press.
[5] Scott, John. 1991. Social Network Analysis. London: Sage.
[6] Wasserman, Stanley, and Faust, Katherine. 1994. Social Network Analysis: Methods and Applications. Cambridge: Cambridge University Press.
[7] The Development of Social Network Analysis Vancouver: Empirical Press.
[8] A.R. Radcliffe-Brown, "On Social Structure," Journal of the Royal Anthropological Institute: 70 (1940): 1–12.
[9] Nadel, SF. 1957. The Theory of Social Structure. London: Cohen and West.
[10] The Networked Individual: A Profile of Barry Wellman (http:/ / www. semioticon. com/ semiotix/ semiotix14/ sem-14-05. html)
[11] Mullins, Nicholas. Theories and Theory Groups in Contemporary American Sociology. New York: Harper and Row, 1973; Tilly, Charles, ed. An Urban World. Boston: Little Brown, 1974; Mark Granovetter, "Introduction for the French Reader," Sociologica 2 (2007): 1–8; Wellman, Barry. 1988. "Structural Analysis: From Method and Metaphor to Theory and Substance." Pp. 19-61 in Social Structures: A Network Approach, edited by Barry Wellman and S.D. Berkowitz. Cambridge: Cambridge University Press.
[12] Mark Granovetter, "Introduction for the French Reader," Sociologica 2 (2007): 1–8; Wellman, Barry. 1988. "Structural Analysis: From Method and Metaphor to Theory and Substance." Pp. 19-61 in Social Structures: A Network Approach, edited by Barry Wellman and S.D. Berkowitz. Cambridge: Cambridge University Press. (see also Scott, 2000 and Freeman, 2004).
[13] Barry Wellman, Wenhong Chen and Dong Weizhen. “Networking Guanxi." Pp. 221–41 in Social Connections in China: Institutions, Culture and the Changing Nature of Guanxi, edited by Thomas Gold, Douglas Guthrie and David Wank. Cambridge University Press, 2002.
[14] Could It Be A Big World After All? (http:/ / www. judithkleinfeld. com/ ar_bigworld. html): Judith Kleinfeld article.
[15] Six Degrees: The Science of a Connected Age, Duncan Watts.
[16] James H. Fowler and Nicholas A. Christakis. 2008. " Dynamic spread of happiness in a large social network: longitudinal analysis over 20 years in the Framingham Heart Study. (http:/ / www. bmj. com/ cgi/ content/ full/ 337/ dec04_2/ a2338)" British Medical Journal. December 4, 2008: doi:10.1136/bmj.a2338. Media account for those who cannot retrieve the original: Happiness: It Really is Contagious (http:/ / www.

npr. org/ templates/ story/ story. php?storyId=) Retrieved December 5, 2008.
[17] Shishkin, Philip (January 27, 2009). "Genes and the Friends You Make" (http:/ / online. wsj. com/ article/ SB123302040874118079. html).
Wall Street Journal. .
[18] Fowler, J. H.; Dawes, CT; Christakis, NA (10 February 2009). "Model of Genetic Variation in Human Social Networks" (http:/ / jhfowler.
ucsd. edu/ genes_and_social_networks. pdf) (PDF). Proceedings of the National Academy of Sciences 106 (6): 1720–1724.
doi:10.1073/pnas.0806746106. PMID 19171900. PMC 2644104. .
[19] The most comprehensive reference is: Wasserman, Stanley, & Faust, Katherine. (1994). Social Networks Analysis: Methods and
Applications. Cambridge: Cambridge University Press. A short, clear basic summary is in Krebs, Valdis. (2000). "The Social Life of Routers."
Internet Protocol Journal, 3 (December): 14–25.
[20] Cohesive.blocking (http:/ / intersci. ss. uci. edu/ wiki/ index. php/ Cohesive_blocking) is the R program for computing structural cohesion
according to the Moody-White (2003) algorithm. This wiki site provides numerous examples and a tutorial for use with R.
[21] Moody, James, and Douglas R. White (2003). "Structural Cohesion and Embeddedness: A Hierarchical Concept of Social Groups."
American Sociological Review 68(1):103–127. Online (http:/ / www2. asanet. org/ journals/ ASRFeb03MoodyWhite. pdf): (PDF file).
[22] Bernie Hogan, Juan-Antonio Carrasco and Barry Wellman, "Visualizing Personal Networks: Working with Participant-Aided Sociograms,"
Field Methods 19 (2), May 2007: 116-144.
[23] USPTO search on published patent applications mentioning “social network” (http:/ / appft. uspto. gov/ netacgi/ nph-Parser?Sect1=PTO2&
Sect2=HITOFF& u=/ netahtml/ PTO/ search-adv. html& r=0& p=1& f=S& l=50& Query=spec/ "social+ network"& d=PG01)
[24] USPTO search on issued patents mentioning “social network” (http:/ / patft. uspto. gov/ netacgi/ nph-Parser?Sect1=PTO2&
Sect2=HITOFF& u=/ netahtml/ PTO/ search-adv. htm& r=0& p=1& f=S& l=50& Query=spec/ "social+ network"& d=PTXT)