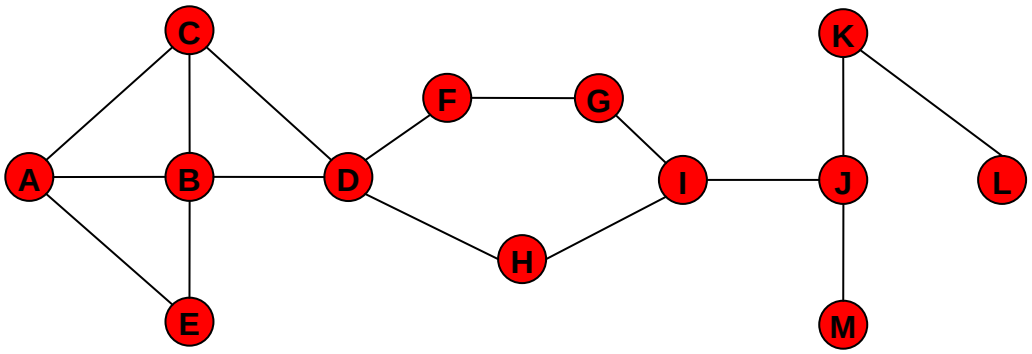
INTRODUCTION TO

THE FORMAL ANALYSIS

OF SOCIAL NETWORKS

USING MATHEMATICA



Luis R. Izquierdo
Robert A. Hanneman

TABLE OF CONTENTS

1 PRELIMINARY NOTE

This document is heavily based on the book "Introduction to Social Network Methods", by Robert A. Hanneman and Mark Riddle (2005), which is freely available online to use and reproduce (with citation). A few sentences have even been copied literally.

Text inserted in light grey boxes refers to slightly more advanced concepts, so you are free to ignore it if the formal analysis of social networks is new to you.

Text inserted in dark grey boxes refers to code that you can run using Mathematica.

Text in this format refers to the output of the code written in the code immediately above.

2 BASIC CONCEPTS AND TERMINOLOGY

The term network has different meanings in different disciplines. In the social sciences, a network is usually defined as a set of actors (or agents, or nodes, or points, or vertices) that may have relationships (or links, or edges, or ties) with one another (Figure 1). Networks can have few or many actors, and one or more kinds of relations between pairs of actors.

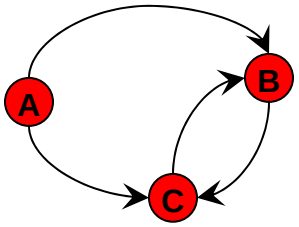


Figure 1. Example of a directed network.

Networks that represent a single type of relation among the actors are called *simplex*, whilst those that represent more than one kind of relation are called *multiplex*. In this document we only deal with simplex networks. When more than one type of relation is present, this can be analysed using different networks, one for each type.

Each tie or relation may be *directed* (i.e. originates in a source actor and reaches a target actor, e.g. the relation "to be a parent of"), or it may be a tie that represents co-occurrence, co-presence, or a bonded-tie between the pair of actors (*undirected*, e.g. the relation "to be a sibling of"). Directed ties are represented with arrows (see Figure 1), and bonded-tie relations are represented with line segments. Directed ties may be reciprocated (A links to B and B links to A); such ties can be represented with a double-headed arrow.

The ties may have different strengths or weights. These strengths may be e.g. binary (representing presence or absence of a tie), signed (representing a negative tie, a positive tie, or no tie); ordinal (representing whether the tie is the strongest, next strongest, etc.); or numerically valued (measured on an interval or ratio scale).

Networks are formally studied in a branch of mathematics called Graph Theory. The formal abstraction called *network* in the social sciences is often named *graph* in Graph Theory, while the term "network" in Graph Theory is reserved for a specific type of graph.

To be precise, a (directed) graph G in graph theory is defined as an ordered pair $G:=(V, A)$ subject to the following conditions:

- V is a set, whose elements are variously referred to as **nodes**, **points**, or **vertices**.
- A is a set of ordered pairs of vertices, called **arcs**, **arrows**, or **directed edges**. An edge $e = (x, y)$ is said to be directed **from** x **to** y, where x is the **tail** of e and y is the **head** of e.

By definition, a network in Graph Theory is a directed graph with weighted edges (Figure 2).

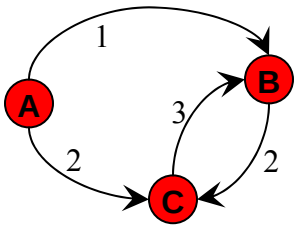


Figure 2. Example of a directed graph with weighted edges.

When a network is drawn, it is sometimes called a graph, but this may cause confusion because, as explained above, the term graph in Graph Theory is the abstract, non-graphical structure. Thus, in Graph Theory, a *graph drawing* is a different concept from the *graph* itself, as there are several ways to structure the graph drawing. All that matters in a graph is which vertices are connected to which others by how many edges (and, potentially, with what weights), and not the exact layout (Figure 3). As a matter of fact, 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.

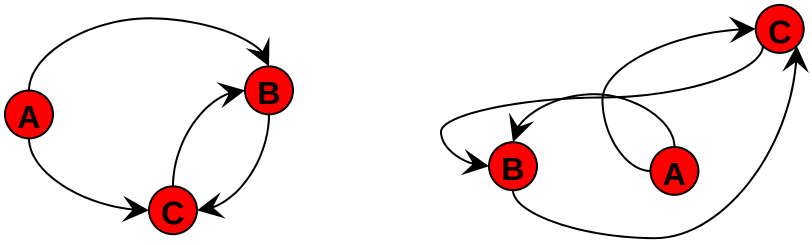
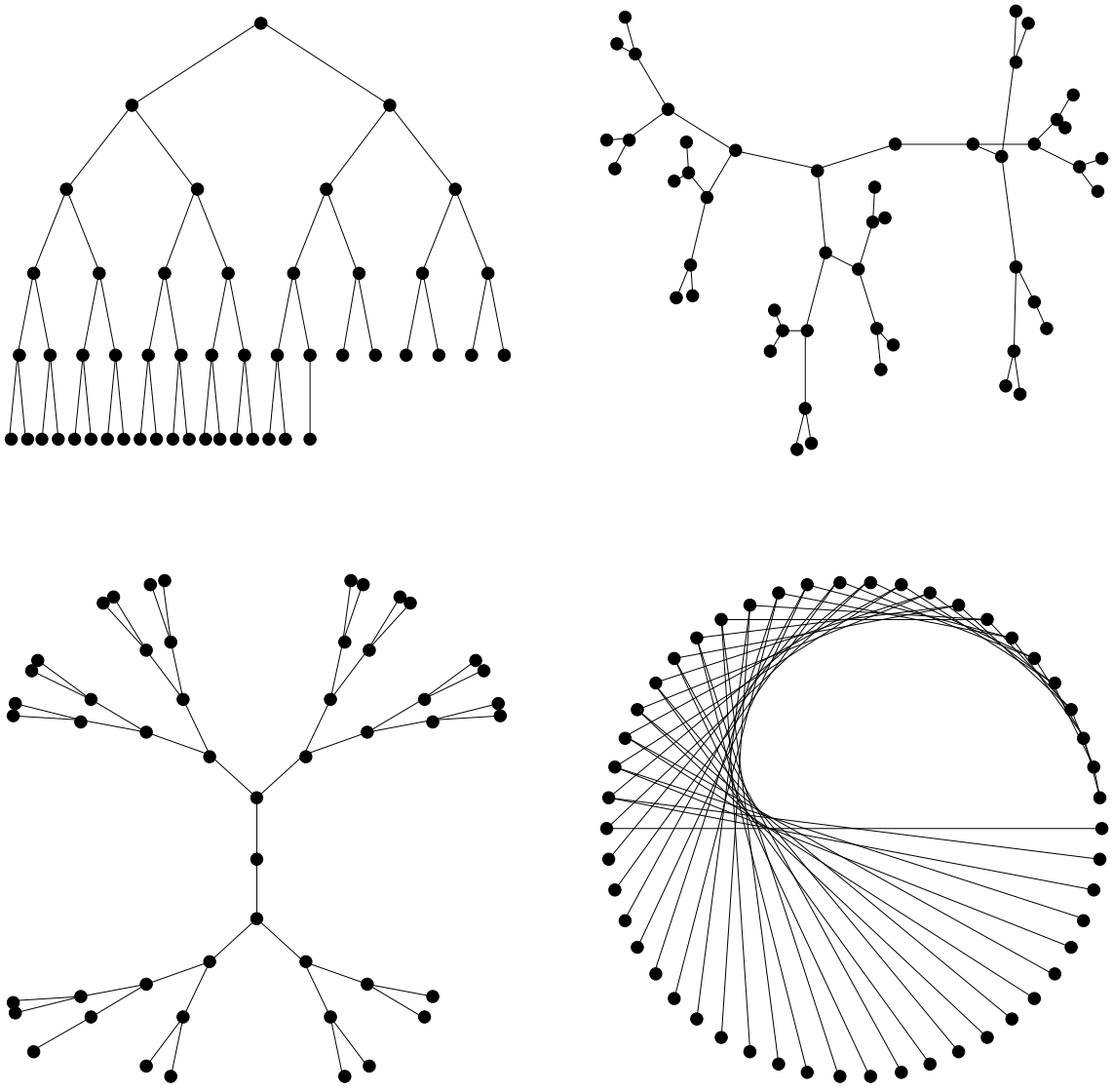


Figure 3. Two different *graph drawings* of the same *graph*.

I do not think that making the distinction between network and graph is critical at all as long as we understand each other, but it is important to be aware of the meaning of the word “graph” in graph theory.

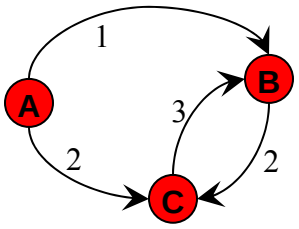
Example

```
Needs["DiscreteMath`Combinatorica`"];

myBinaryTree=CompleteBinaryTree[50];
g1=ShowGraph[myBinaryTree]
g2=ShowGraph[SpringEmbedding[myBinaryTree,200,0.05]];
g4=ShowGraph[CircularEmbedding[myBinaryTree]];
g3=ShowGraph[RadialEmbedding[myBinaryTree]];
Show[GraphicsArray[{{g1,g2},{g3,g4}}]];
```



3 REPRESENTING SOCIAL NETWORK DATA

The two most common ways of representing social network data are by *drawing* the network and by using *matrices*. The most common form of matrix in social network analysis is a square matrix with as many rows (and columns) as actors in the data set. The cells of the matrix record information about the ties between each pair of actors (e.g. their weights). These matrices are read: *Row* links to *Column*.

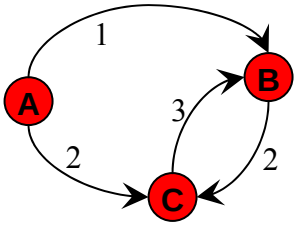


	A	B	C
A	0	1	2
B	0	0	2
C	0	3	0

Figure 4. Two ways of representing a directed weighted graph.

The simplest and most common matrix is binary. That is, if a tie is present, a 1 is entered in a cell; if there is no tie, a 0 is entered. This kind of matrix is called the *adjacency matrix*.

The *adjacency matrix* is extremely useful to conduct various formal analyses of graphs. In particular, note that the adjacency matrix *AdjMatrix* tells us how many paths of length 1 there are from each actor to each other actor. In general, it can be shown that the powers of the adjacency matrix, $AdjMatrix^n$, give the number of walks of length n from each actor to each other actor.



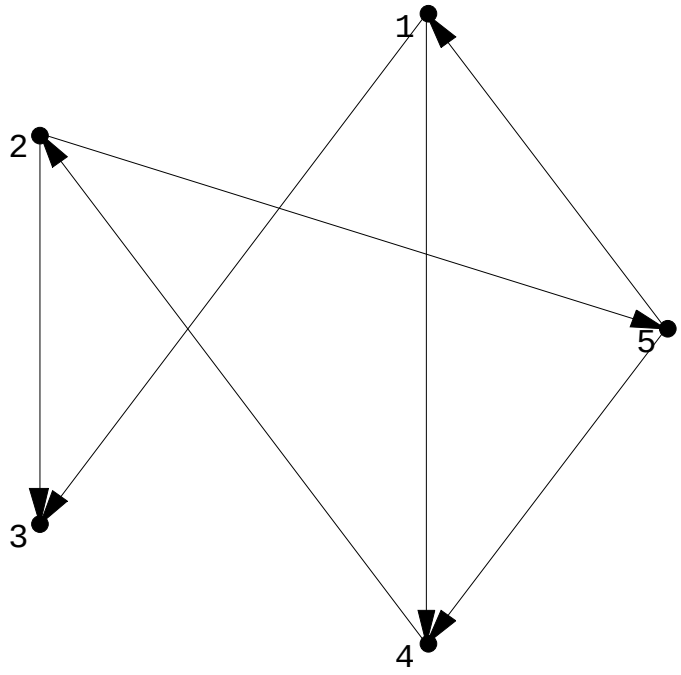
$$AdjMatrix = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \qquad AdjMatrix^2 = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$AdjMatrix^3 = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \qquad AdjMatrix^4 = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Figure 5. A network, and some powers of its adjacency matrix.

Example

```
Needs["DiscreteMath`Combinatorica`"];
SeedRandom[2];

myGraph=RandomGraph[5,0.3,Type→Directed];
ShowGraph[myGraph,
        VertexNumber→True,TextStyle→{FontSize→16},VertexNumberPosition→{-0.01,-0.02}];
```




```
myAdjMatrix=ToAdjacencyMatrix[myGraph];
Table[MatrixForm[MatrixPower[myAdjMatrix,i]],{i,5}]
```

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 2 & 0 \end{pmatrix}$$

4 BASIC PROPERTIES OF NETWORKS AND ACTORS

4.1 Size of a network

The *size* of a network can be determined in terms of the number of nodes of the network (as stated by Hanneman (2005)) or, alternatively, as the number of edges in the network (as written in http://en.wikipedia.org/wiki/Glossary_of_graph_theory). Size in terms of nodes can be critical for the structure of social relations because of the limited resources that each actor may have for building and maintaining ties.

In the network shown in Figure 3: Number of nodes = 3. Number of edges = 4.

```
Print["Number of nodes: ", V[myGraph]];
Print["Number of edges: ", M[myGraph]];
```

```
Number of nodes:  5
Number of edges:  7
```

4.2 Density of a network

The density of a network is the number of ties in the network expressed as a proportion of the number of all possible ties, i.e. the number of actual ties in the network divided by the number of all the ties that could be present. In a directed binary network of size n, the number of possible ties is $n \times (n - 1)$. In an undirected binary network of size n, the number of possible ties is $n \times (n - 1) / 2$. The density of a network may give insights into phenomena such as the speed at which information diffuses among the nodes, or the extent to which actors have high levels of social capital and/or social constraint. It is also sometimes used as a measure of connectivity of the network.

In the network shown in Figure 3: Density = 4 / 6.

```
If[UndirectedQ[myGraph],
  Print["Density: ", 2 M[myGraph] / (V[myGraph] (V[myGraph]-1))],
  Print["Density: ", M[myGraph] / (V[myGraph] (V[myGraph]-1))]
  ];
```

Density: $\frac{7}{20}$

4.3 Degree of actors

In an undirected network, the *degree of a node* is the number of links that such a node has. In directed networks, we have to distinguish between incoming links (*in-degree*) and outgoing links (*out-degree*). In a directed network, statistics on the rows of the adjacency matrix tell us about the role that each actor plays as a source of ties (e.g. the sum of the elements in its row is its out-degree), whereas statistics on the columns tell us about its role as sink of links (e.g. the sum of the elements in its column is its in-degree). The out-degree of an actor is often a measure of how influential the actor may be.

In the network shown in Figure 3:
In-degrees: A = 0; B = 2; C = 2. Out-degrees: A = 2; B = 1; C = 1.

```
Print["The in-degree of each node in the network is: ", InDegree[myGraph]];
```

The in-degree of each node in the network is: {1,1,2,2,1}

```
Print["The out-degree of each node in the network is: ", OutDegree[myGraph]];
```

The out-degree of each node in the network is: {2,2,0,1,2}

A *degree sequence* is a list of degrees of a graph in non-increasing order (e.g. $d_1 \geq d_2 \geq \ldots \geq d_n$). A sequence of non-increasing integers is *realisable* if it is a degree sequence of some graph.

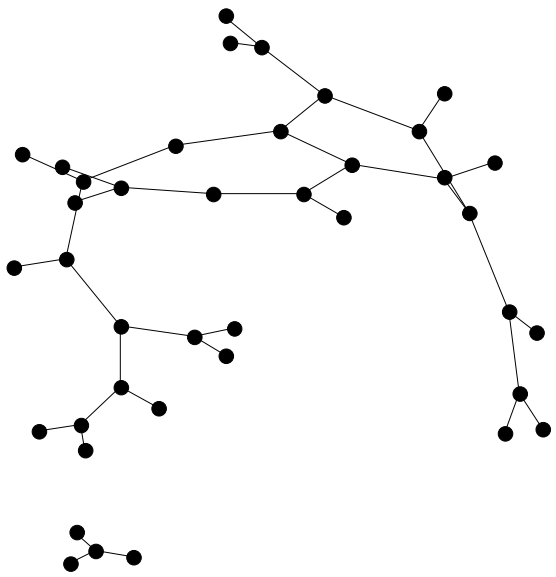
```
DegreeSequence[MakeUndirected[myGraph]]
```

{3,3,3,3,2}

```
myList=DegreeSequence[CompleteBinaryTree[40]]
```

{3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,2,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}

```
ShowGraph[SpringEmbedding[
        RealizeDegreeSequence[myList]
        , 100, 0.05 ]]
```



5 SOCIAL DISTANCE AND RELATED CONCEPTS

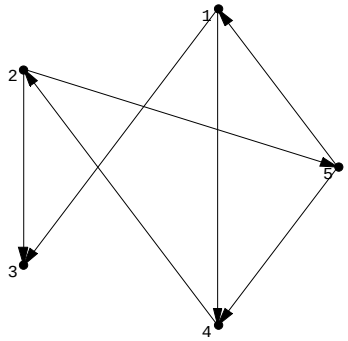
5.1 Distance between actors

The properties of the network that we have examined so far deal primarily with immediate adjacency (the actor's immediate social neighbours). However, the connections of an actor's social neighbours can be very important, even if the actor is not directly connected to them (e.g. think of the importance of having "well-connected friends" in certain environments). In other words, sometimes being a "friend of a friend" may be quite consequential.

To capture this aspect of how individuals are embedded in networks, one approach is to examine *how far* (in terms of *social distance*) an actor is from others. The *distance* between two actors is the minimum number of edges that takes to go from one to another[1]. This is also known as the *geodesic distance*. Those actors who are closer to more others may be able to exert more power than those who are more distant. We will study this in detail further on.

If two actors are adjacent, the distance between them is 1 (i.e. it takes one step, or edge, to go from one to the other). If A links to B, and B links to C (and A does not link to C), then actors A and C are at a distance of 2. Sometimes we are also interested in studying the various ways that two actors, which are at a given distance, can be connected; multiple connections may indicate a stronger relation between two actors than a single connection.

```
Needs["DiscreteMath`Combinatorica`"];
SeedRandom[2];
myGraph=RandomGraph[5,0.3,Type→Directed];
ShowGraph[myGraph,
      VertexNumber→True,TextStyle→{FontSize→16},VertexNumberPosition→{-0.01,-0.02}];
Print["All pairs distances: ",MatrixForm[AllPairsShortestPath[myGraph]]];
```



All pairs distances:

$$\begin{pmatrix} 0 & 2 & 1 & 1 & 3 \\ 2 & 0 & 1 & 2 & 1 \\ \infty & \infty & 0 & \infty & \infty \\ 3 & 1 & 2 & 0 & 2 \\ 1 & 2 & 2 & 1 & 0 \end{pmatrix}$$

[1] Formally, the distance between two vertices is the length of a shortest *path* between them, but the definition of *path* is still to come.

5.2 Walks

The most general form of connection between two actors in a graph is called a *walk*. A walk is a sequence of actors and relations that begins and ends with actors[2]. A *closed walk* is one where the beginning and end point of the walk are the same actor. Walks are unrestricted: a walk can involve the same actor or the same relation multiple times. The *length of a walk* is the number of edges that it uses.

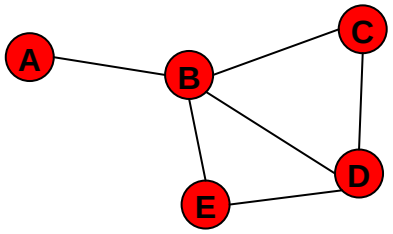
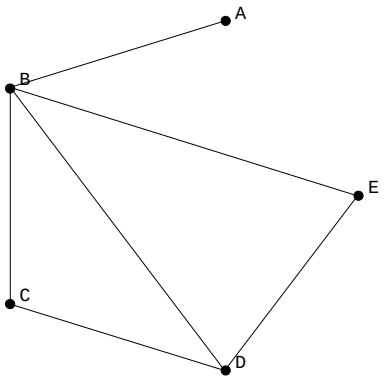


Figure 6. An undirected graph.

Some examples of walks between A and C in the graph represented in Figure 6 are:

{A, B, C}	length = 2;
{A, B, D, C}	length = 3;
{A, B, E, D, C}	length = 4;
{A, B, D, B, C}	length = 4;

```
adjMatrixFig6={{0,1,0,0,0},{1,0,1,1,1},{0,1,0,1,0},{0,1,1,0,1},{0,1,0,1,0}};
graphFig6=SetVertexLabels[FromAdjacencyMatrix[adjMatrixFig6],{A,B,C,D,"E"}];
ShowGraph[graphFig6, TextStyle→{FontSize→16}];
```




```
Table[  Print["Number of walks of length ", i, " between nodes: ",
        MatrixForm[MatrixPower[myAdjMatrix, i]]], {i, 4}];
```

[2] Formally, a *walk* is an alternating sequence of vertices and edges, beginning and ending with a vertex, in which each vertex is incident to the two edges that precede and follow it in the sequence, and the vertices that precede and follow an edge are the end vertices of that edge.

Number of walks of length 1 between nodes:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Number of walks of length 2 between nodes:

$$\begin{pmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 4 & 1 & 2 & 1 \\ 1 & 1 & 2 & 1 & 2 \\ 1 & 2 & 1 & 3 & 1 \\ 1 & 1 & 2 & 1 & 2 \end{pmatrix}$$

Number of walks of length 3 between nodes:

$$\begin{pmatrix} 0 & 4 & 1 & 2 & 1 \\ 4 & 4 & 6 & 6 & 6 \\ 1 & 6 & 2 & 5 & 2 \\ 2 & 6 & 5 & 4 & 5 \\ 1 & 6 & 2 & 5 & 2 \end{pmatrix}$$

Number of walks of length 4 between nodes:

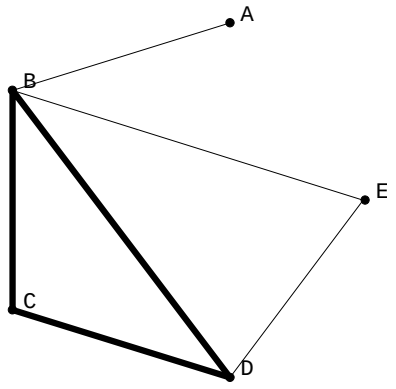
$$\begin{pmatrix} 4 & 4 & 6 & 6 & 6 \\ 4 & 22 & 10 & 16 & 10 \\ 6 & 10 & 11 & 10 & 11 \\ 6 & 16 & 10 & 16 & 10 \\ 6 & 10 & 11 & 10 & 11 \end{pmatrix}$$

In directed networks, in addition to walks we can also define *semi-walks*, which are walks of the underlying undirected network (i.e. ignoring the directionality of the connections).

5.3 Cycles

A *cycle* is a specially restricted walk that is often used in algorithms examining the neighbourhoods of actors (i.e. the points adjacent to a particular node). A cycle is a closed walk of 3 or more actors, all of whom are distinct, except for the origin/destination actor. There are no cycles beginning and ending with A in Figure 6, but there are 3 beginning and ending with actor B ({B, D, C, B}; {B, E, D, B}; {B, C, D, E, B}).

```
?ExtractCycles (* ExtractCycles[g] gives a maximal list of edge-disjoint cycles in graph g. *)
ShowGraph[Highlight[graphFig6,ExtractCycles[graphFig6]],TextStyle→ {FontSize→ 16}]
```



5.4 Trails

Sometimes it may be useful to study only those walks that do not re-use relations. A *trail* between two actors is any walk that includes any given relation at most once. (The same actors, however, can be part of a trail multiple times). The *length of a trail* is the number of relations in it. All trails are walks, but not all walks are trails. If the trail begins and ends with the same actor, it is called a *closed trail*. In Figure 6 there are a number of trails from A to C. Excluded are tracings like {A, B, D, B, C} (which is a walk, but is not a trail because the relation BD is used more than once).

In directed networks, in addition to trails we can also define *semi-trails*, which are trails of the underlying undirected network (i.e. ignoring the directionality of the connections).

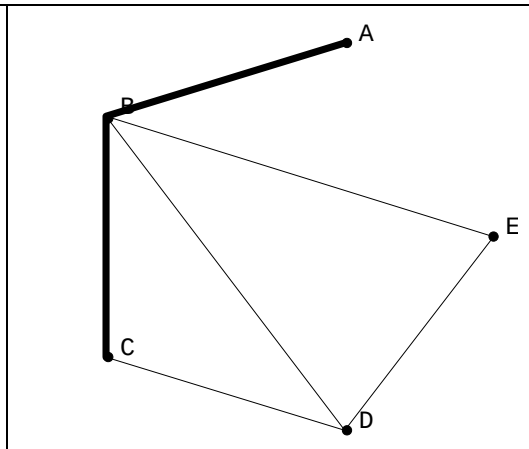
5.5 Paths

Perhaps the most useful definition of a connection between two actors (or between an actor and itself) is a *path*. A *path* is a walk in which each actor (and therefore each relation) in the graph may be used at most once[3]. The single exception to this is a *closed path*, which begins and ends with the same actor. All paths are trails and walks, but not all walks and all trails are paths. In Figure 6, there are a limited number of paths connecting A and C: {A, B, C}; {A, B, D, C}; {A, B, E, D, C}.

The *length of a path* is the number of relations in it. The length of a shortest path between two actors is the geodesic distance between them. Thus, the geodesic distance between A and C in the graph of Figure 6 is 2.

```
Print["Shortest path between A (node 1) and C (node 3): ", ShortestPath[graphFig6,1,3]];
ShowGraph[Highlight[graphFig6, ShortestPath[graphFig6, 1, 3]]]
```

Shortest path between
A (node 1) and C (node 3):
{1,2,3}



[3] Some authors use a slightly different terminology: they use the term "path" to refer to a walk and they use the term "simple path" to refer to a path. Nowadays, when stated without any qualification, a path is usually defined to be *simple*, meaning that every vertex is incident to at most two edges (http://en.wikipedia.org/wiki/Glossary_of_graph_theory#Walks).

When we have measures of the strengths of ties (e.g. the dollar volume of trade between nations), the "distance" between two actors is usually defined as the strength of the weakest (e.g. least costly) path between them. In directed networks, in addition to paths we can also define *semi-paths*, which are paths of the underlying undirected network (i.e. ignoring the directionality of the connections).

5.6 Eccentricity of actors

For each actor, we could calculate the distribution of its geodesic distances to the other actors. An actor's largest geodesic distance is called its *eccentricity* - a measure of how far an actor is from the furthest other. If two nodes are not reachable from each other (i.e. the network is disconnected), their geodesic distance is infinite.

```
Print["The eccentricity of each node in the network is: ", Eccentricity[graphFig6]];
```

```
The eccentricity of each node in the network is:  {2,1,2,2,2}
```

5.7 Diameter and radius of a network

The *diameter* of a network is the maximum eccentricity over all the actors of the network, i.e. the largest geodesic distance in the (connected) network (if the network is not connected the largest distance is infinity). The diameter of a network gives the number of steps that are sufficient to go from any node to any other node (i.e. the minimum path length that can connect any pair of nodes in the network). The diameter is sometimes used as a measure of connectivity of a network.

```
Print["The diameter of the network is: ", Diameter[graphFig6]];
```

```
The diameter of the network is:  2
```

The *radius* of a network is the minimum eccentricity over all the actors of the network. Trivially, in an graph G, $\text{diam}(G) \leq 2\,\text{rad}(G)$. When there is more than one component in a network, its diameter and its radius are defined to be infinity.

Vertices with maximum eccentricity are called *peripheral vertex*. Vertices of minimum eccentricity form the *centre*.

```
Print["The radius of the network is: ", Radius[graphFig6]];
```

```
The radius of the network is:  1
```

```
Print["The peripheral vertices are: ", Position[Eccentricity[graphFig6], Diameter[graphFig6]]];
```

```
The peripheral vertices are:  {{1},{3},{4},{5}}
```

```
Print["The nodes in the centre of the network are: ", GraphCenter[graphFig6]];
```

```
The nodes in the centre of the network are:  {2}
```

6 CONNECTION AND CONNECTIVITY

6.1 Reachability

Reachability between nodes is established by the existence of a path between the nodes. In simpler words, an actor is "reachable" by another if there exists a set of connections by which we can go from the source to the target actor, regardless of how many others fall between them. If the data are directed, it is possible that actor A can reach actor B, but that actor B cannot reach actor A (e.g. hyperlinks in web pages).

In the network shown in Figure 3: A is not reachable from any node; B is reachable from A and C; and C is reachable from A and B. In the network shown in Figure 6, every node is reachable from every other node; when this is the case we say that the network is *connected* (see below).

```
Print["Is the graph in fig. 6 connected? ", ConnectedQ[graphFig6]];
```

```
Is the graph in fig. 6 connected?  True
```

6.2 Connectivity of a network in Graph Theory

Adjacency tells us about direct connections between actors. Reachability tells us about whether actors are connected or not allowing for pathways of any length. Connectivity is a property of a network (not of its individual actors) that extends the concept of adjacency. If it is possible to establish a path from any actor to any other actor of a network (e.g. every actor is reachable by every other one), the network is said to be *connected*; otherwise the network is *disconnected*. The network represented in Figure 6 is connected. The two networks represented at the bottom of Figure 7 are disconnected.

A *component* is a (maximal) set of nodes that are connected (i.e. all nodes in the subgraph are reachable from all other nodes in the subgraph). The network represented at the bottom-right corner of Figure 7 has 2 components. The network represented at the bottom-left corner of Figure 7 has 3 components.

In the context of directed networks, we distinguish between *strongly connected* and *weakly connected* networks (and components). A directed network is *strongly connected* if every node is reachable from every other node following the directions of the edges. A directed network is *weakly connected* if its underlying undirected graph is connected. The network represented in Figure 3 is strongly disconnected, but weakly connected.

```
Print["Is my graph strongly connected? ", ConnectedQ[myGraph, Strong]];
```

```
Is my graph strongly connected?  False
```

```
Print["The strongly connected components in my graph are: ",
        StronglyConnectedComponents[myGraph]];
```

```
The strongly connected components in my graph are:  {{1,2,4,5},{3}}
```

```
Print["Is my graph weakly connected? ", ConnectedQ[myGraph, Weak]];
```

Is my graph weakly connected? True

```
Print["The weakly connected components in my graph are: ",
        WeaklyConnectedComponents[myGraph]];
```

The weakly connected components in my graph are: {{1,2,3,4,5}}

If it is always possible to establish a path from any actor (i.e. vertex) to every other one even after removing any $k - 1$ vertices, then the graph is said to be *k-connected* in vertices. This concept is the base to define vertex connectivity.

```
?VertexConnectivity
Print["The vertex-connectivity of graph in fig. 6 is: ", VertexConnectivity[graphFig6]];
```

VertexConnectivity[g] gives the minimum number of vertices whose deletion from graph g disconnects it. VertexConnectivity[g, Cut] gives a set of vertices of minimum size, whose removal disconnects the graph.

The vertex-connectivity of graph in fig. 6 is: 1

A *cut vertex*, or *articulation point*, is a vertex whose removal disconnects a graph. A *cut set*, or *vertex cut,* or *separating set*, is a set of vertices whose removal disconnects the graph.

```
?ArticulationVertices
Print["The articulation vertices of graph in fig. 6 are: ", ArticulationVertices[graphFig6]];
```

ArticulationVertices[g] gives a list of all articulation vertices in graph g. These are vertices whose removal will disconnect the graph.

The articulation vertices of graph in fig. 6 are: {2}

If it is always possible to establish a path from any actor (i.e. vertex) to every other one even after removing any $k - 1$ edges, then the graph is said to be *k-connected* in edges. This concept is the base to define edge connectivity.

```
?EdgeConnectivity
Print["The edge-connectivity of graph in fig. 6 is: ", EdgeConnectivity[graphFig6]];
```

EdgeConnectivity[g] gives the minimum number of edges whose deletion from graph g disconnects it. EdgeConnectivity[g, Cut] gives a set of edges of minimum size whose deletion disconnects the graph.

The edge-connectivity of graph in fig. 6 is: 1

A *bridge*, or *cut edge* or *isthmus*, is an edge whose removal disconnects a graph. A *disconnecting set* is a set of edges whose removal disconnects a graph (and it therefore increases the number of *components*).

```
?Bridges
Print["The bridges of graph in fig. 6 are: ", Bridges[graphFig6]];
```

Bridges[g] gives a list of the bridges of graph g, where each bridge is an edge whose removal disconnects the graph.

The bridges of graph in fig. 6 are: {{1,2}}

6.3 Connectivity of a network in normal speech

Note that *connectivity* (and *k-connectivity*) in Graph Theory is a binary concept (i.e. a network is either connected or not). To quantify connectivity, analysts tend to use the concept of edge / node connectivity, which is the minimum number of edges / nodes, whose removal disconnects the graph, i.e. what is the maximum value of *k* such that the network is *k*-connected?

However, even when considering networks with the same edge connectivity and node connectivity, one sometimes hears (or wants to make!) statements like: "this network is better connected than this other one". What does that mean? Connectivity when used in this sense is usually a vague concept which is not formally defined, but there are several metrics in Graph Theory that can be useful to pin it down.

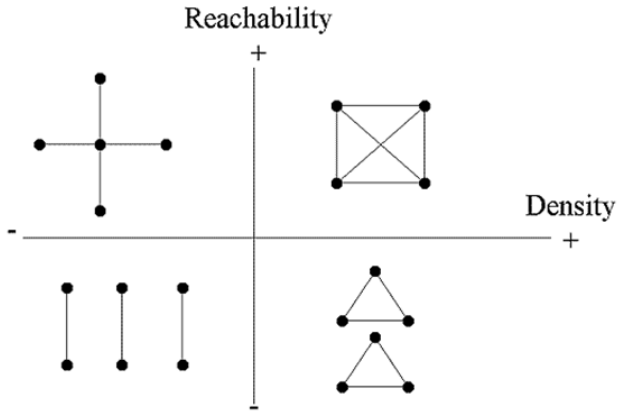


Figure 7. Different types of networks as a function of reachability and density. Source: Janssen et al. (2006).

The simplest graph theoretical concept that can be used as a measure of "connectivity" is the density of the network. A very dense network is generally considered to be well connected. Another aspect of "connectivity" is "reachability" (understood, somewhat informally for networks, as the extent to which all the nodes in the network are accessible to each other). As Janssen et al. point out (2006) these two aspects of "connectivity" – i.e. density and "reachability" – are not completely independent, and one could say that high density normally implies high reachability. They are, however, not the same, and it is possible to have networks with both high density and low reachability if there is a high level of clustering, i.e. the links are distributed only within, and never between, isolated clusters (see Figure 7).

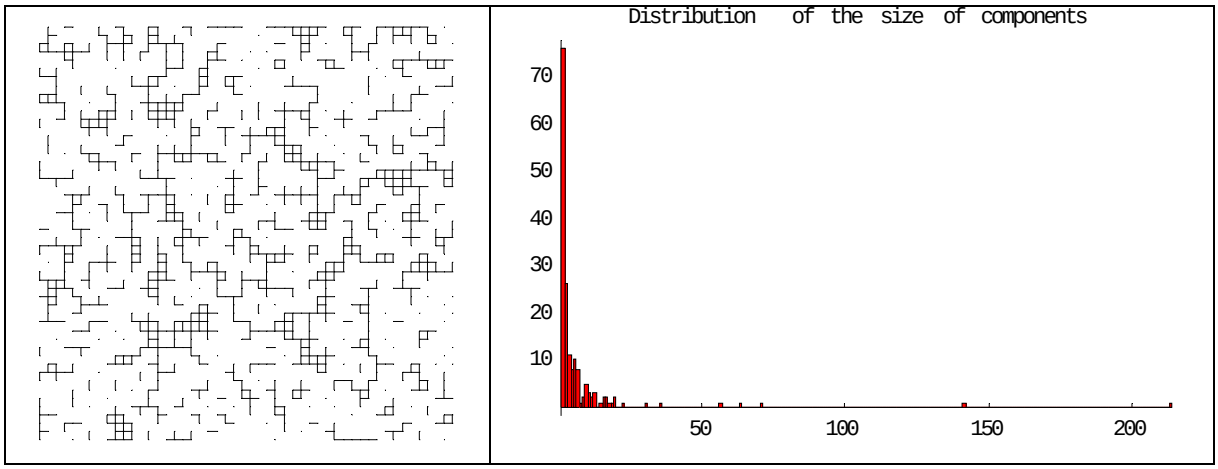
To quantify *reachability of a network* (which so far we have used only informally, since it does not have a precise definition in Graph Theory), Janssen et al. (2006) suggest using the network diameter (see above), and/or the size of the largest component (see section

"Connectivity of a network in Graph Theory" above). A short diameter implies that it is possible to move through the whole network in just a few steps. Similarly, if the largest component contains a large fraction of all the nodes in the network, then there is a high probability that any two nodes are interconnected.

Other metrics that can be used to quantify the connectivity of a network (in addition to density and reachability) are:

- *Node connectivity*: The minimum number of nodes that have to be removed to disconnect the network.
- *Edge connectivity*: The minimum number of edges that have to be removed to disconnect the network.
- Number of *bridges*: a bridge is an edge whose removal disconnects a graph.
- Distribution (number and size) of *connected components*.

```
Needs["Graphics`Graphics`"];
ShowGraph[myFragmentedGrid =
        InduceSubgraph[GridGraph[50,50],RandomSubset[2500]],VertexStyle→Disk[0]];
c = Map[Length,ConnectedComponents[myFragmentedGrid]];
Histogram[c, HistogramCategories→Range[0,Max[c]+1],
        PlotLabel→"Distribution of the size of components"];
```



7 LOCAL STRUCTURES IN NETWORKS

So far we have looked mainly at tools for examining ways in which individuals are connected, and the distances between them. In this section we look at the same issue of connection but this time our focus is the social structure, rather than the individual: here we adopt a slightly more "macro" perspective that focuses on the local structures within which individual actors are embedded.

The smallest social structure in which an individual can be embedded is a *dyad* (i.e. a pair of actors). For binary ties (present or absent), there are two possibilities for each pair of

nodes in the network - either they have a tie, or they don't. If we are considering a directed relation, there are three kinds of dyads (no tie, one links to the other but not vice versa, or they both link to each other). A potentially interesting analysis is to study the extent to which a population is characterized by reciprocated ties; this may tell us about the degree of cohesion, trust, and social capital that is present (see *reciprocity* below).

Another form of social structure is a triad, which is formed by three actors. Triads allow for a much wider range of possible sets of relations (with directed data, there are 64 possible types of relations among 3 specific actors), including relationships that exhibit hierarchy, equality, and the formation of exclusive groups (e.g. where two actors connect, and exclude the third). A potentially interesting analysis is to study the proportion of triads that are "transitive" (see *transitivity* below).

Other examples of social structures embedded in networks that we will consider here are cliques, N-cliques, and N-clans. In this section I explain some metrics of Graph Theory that may be useful to analyse these local structures and the way actors are embedded in a network.

7.1 Dyads and reciprocity

Arguably, a network that has a predominance of null or reciprocated ties over asymmetric connections may be more "equal" or "stable" than one with a predominance of asymmetric connections (which may be more of a hierarchy).

There are two different approaches to quantifying the degree of reciprocity in a population. One approach is to *focus on the dyads*, and calculate the proportion of pairs that have a reciprocated tie between them. In the network shown in Figure 8, this approach would yield a reciprocity rate of 1/3. More commonly, however, analysts are concerned with the ratio of the number of pairs with a reciprocated tie relative to the number of pairs with any tie. In large populations it is often the case that most actors have no direct ties to most other actors, and it may be more sensible to focus on the degree of reciprocity among pairs that have any ties. In the network shown in Figure 8, this would yield a reciprocity rate of 1/2.

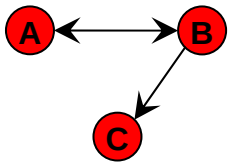


Figure 8. A directed graph.

A second (equivalent) approach consists in *focusing on the relations*, rather than on the actors. The question now is: what percentage of all possible ties are part of reciprocated structures? In the network shown in Figure 8, two such ties (A to B, and B to A) form part of a reciprocated structure among the six possible ties (AB, BA, AC, CA, BC, CB), yielding a reciprocity rate of 1/3. This calculation is equivalent to the one conducted when we focus on the dyads and we consider all possible pairs. Again, analysts usually consider, instead, the number of ties that are involved in reciprocal relations relative to

the total number of actual ties (rather than possible ties). In that case, the reciprocity rate would be 2/3. There is a one-to-one relationship between this number and the one calculated when focusing on the dyads considering only the pairs with at least one tie. Assuming the latter is x/y, the former is $2x/(y+x)$.

```
graphFig8=EmptyGraph[3,Type->Directed];
graphFig8=AddEdges[graphFig8,{{1,2},{2,3},{2,1}}];

Print["Reciprocated relations relative to all possible relations: ",
        ReciprocatedEdges[graphFig8]/(V[graphFig8]*(V[graphFig8]-1))];
Print["Reciprocated relations relative to existing relations: ",
        ReciprocatedEdges[graphFig8]/M[graphFig8]];
```

```
Reciprocated relations relative to all possible relations: 1/3
Reciprocated relations relative to existing relations: 2/3
```

7.2 Triads and transitivity

With undirected data, there are four possible types of triadic relations (no ties, one tie, two ties, or all three ties). Counts of the relative prevalence of these four types of relations across all possible triples can give a good sense of the extent to which a population is characterized by "isolation," "couples only," "structural holes" (i.e. where one actor is connected to two others, who are not connected to each other), or "clusters".

With directed data, there are actually 16 possible types of relations among 3 actors, including relationships that suggest hierarchy, equality, and the formation of exclusive groups. To identify the frequency of each of these relations we may wish to conduct a "triad census" for each actor, and for the network as a whole. In particular, we may be interested in the proportion of triads that are "transitive" (that is, display a type of balance where, if A directs a tie to B, and B directs a tie to C, then A also directs a tie to C).

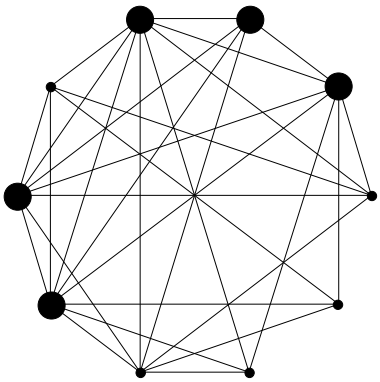
```
?TransitiveQ
?TransitiveClosure
?TransitiveReduction
```

```
TransitiveQ[g] yields True if graph g defines a transitive relation.
TransitiveClosure[g] finds the transitive closure of graph g, the supergraph of g that contains edge {x, y} if and only if there is a path from x to y.
TransitiveReduction[g] finds a smallest graph that has the same transitive closure as g.
```

7.3 Cliques

Every member of a group of people in a social clique knows everybody else. A *clique* is a subset of the vertices such that every pair of vertices in the subset is connected by an edge.

```
SeedRandom[0]
myRandomGraph=RandomGraph[10, 0.7]
ShowGraph[Highlight[myRandomGraph,{MaximumClique[myRandomGraph]}]]
```



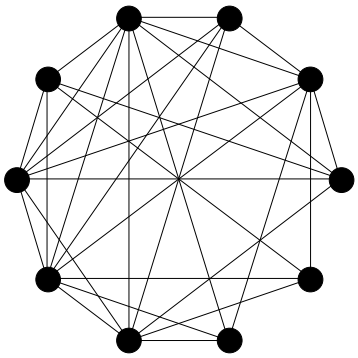
```
Print["Any subset of the vertices of a complete graph forms a clique: ",
        CliqueQ[CompleteGraph[10],RandomSubset[10]]];
```

Any subset of the vertices of a complete graph forms a clique: True

7.4 N-Cliques

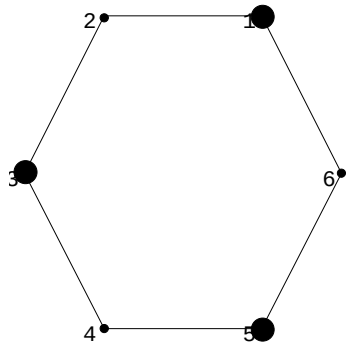
The strict definition of clique (i.e. everyone connected to everyone else) may be too strong for some purposes. A more general approach is to define an actor as a member of a clique if it is connected to every other member of the clique at a distance no greater than a given number. This approach to defining sub-structures is called *N-clique*, where N stands for the length of the path allowed to make a connection to all other members.

```
ShowGraph[Highlight[myRandomGraph,{MaximumNClique[myRandomGraph,2]}]];
```



N-cliques with N > 1 may exhibit the potentially undesirable property that some members of a clique may be connected to other members of the clique by actors who are not themselves members of the clique. The definition of *N-clan* (see below) overcomes this potential problem.

```
ShowGraph[Highlight[Cycle[6],{1,3,5}], VertexNumber->True];
Print["Do nodes 1, 3, and 5 form a 2-Clique? ", NCliqueQ[Cycle[6],2,{1,3,5}]];
Print["Do nodes 1, 2, 3, and 5 form a 2-Clique?", NCliqueQ[Cycle[6],2,{1,2,3,5}]];
```

Do nodes 1, 3, and 5 form a 2-Clique? True Do nodes 1, 2, 3, and 5 form a 2-Clique? False	

7.5 N-Clans

An N-clan is an N-clique where all ties among members of the N-clique occur through members of the N-clique.

```
Print["Do nodes 1, 3, and 5 form a 2-Clan? ", NClanQ[Cycle[6], 2, {1,3,5}]];
Print["2-Clans of maximum size in a cycle with 6 nodes: ", MaximumNClans[Cycle[6], 2]];
```

```
Do nodes 1, 3, and 5 form a 2-Clan?  False
2-Clans of maximum size in a cycle with 6 nodes:  {{1,2,3},{1,2,6},{1,5,6},{2,3,4},{3,4,5},{4,5,6}}
```

7.6 Clustering

7.6.1 Motivation: Small-world networks

This subsection is copied almost literally from Hanneman, R. A. (2005).

Watts (1999) and many others have noted that in large real-world networks (e.g. in the Internet) there is often a structural pattern that seems somewhat paradoxical.

On the one hand, the average distance between any two nodes is relatively short. The "6-degrees" of distance phenomenon is an example of this (see http://en.wikipedia.org/wiki/Six_degrees). Most of the nodes in even very large networks may be fairly close to one another. To be precise, the average geodesic distance between pairs of actors in large empirical networks is often much shorter than in random graphs of the same size.

On the other hand, most actors live in local neighbourhoods where most others are also connected to one another. In other words, in many large networks, a very large proportion of the total number of ties are highly "clustered" into local neighbourhoods. To be precise, the density in local neighbourhoods of many large networks tends to be much higher than we would expect for a random graph of the same size.

To summarise in an informal way, most of the people we know may also know each other - seeming to locate us in a very narrow social world. Yet, at the same time, we can be at quite short distances to vast numbers of people that we don't know at all (e.g. I found out the other day that I'm just two degrees of separation away from Clinton! ☺).

The *small world* phenomena - a combination of short average path lengths over the entire graph coupled with a strong degree of "clique-like" local neighbourhoods - seems to have evolved independently in a wide range of large empirical networks.

We already know how to calculate average path lengths. In the following subsection we will learn how to quantify clustering.

7.6.2 Quantifying clustering

Informally, the *clustering coefficient* is a measure of the extent to which the friends of my friends are my friends. More precisely, the *clustering coefficient of a node* is the ratio of existing links connecting the node's neighbours to each other, to the maximum possible number of such links. For nodes with fewer than two neighbours the clustering coefficient is undefined.

The clustering coefficient of a node A is 1 if every neighbour connected to A is also connected to every other node within the neighbourhood of A, and 0 if no node that is connected to A connects to any other node that is connected to A.

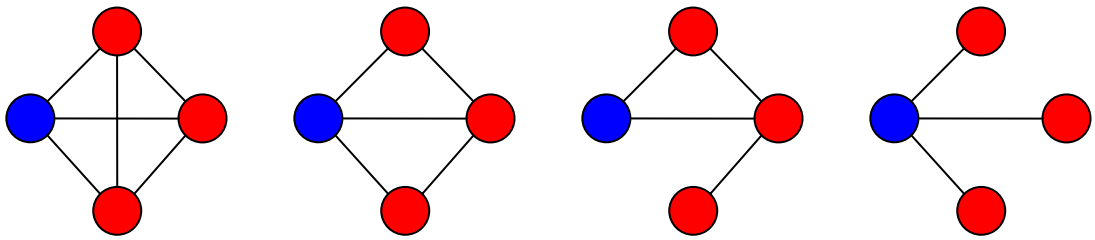


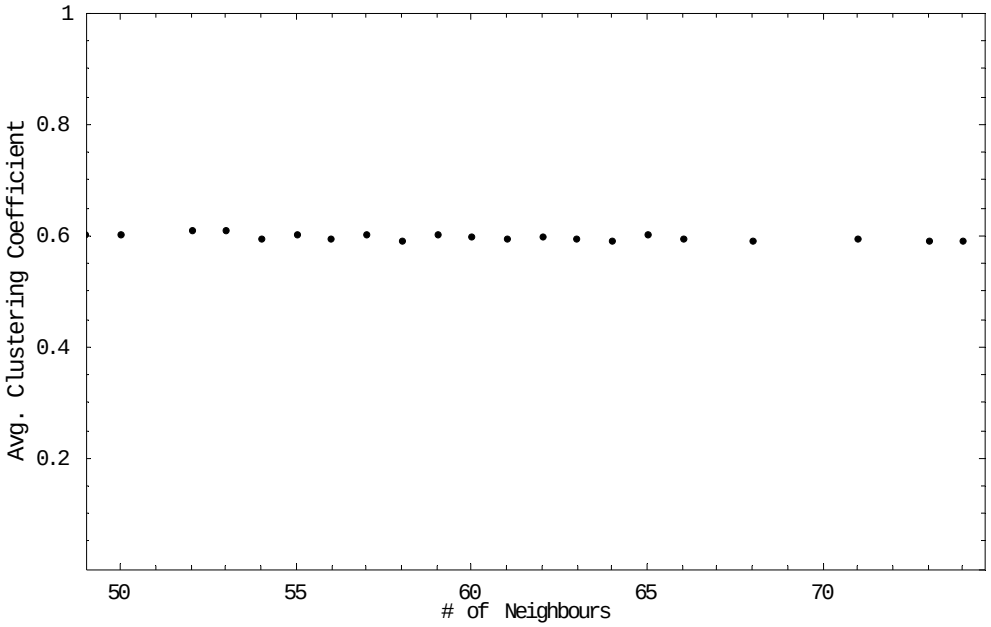
Figure 9. Clustering coefficient of the blue node in various undirected networks (from left to right): 3/3, 2/3, 2/2, 0/3.

```
Print["The clustering coeff. of the blue node in the second graph of fig. 9 is: ",
        Clustering[1,DeleteEdge[CompleteGraph[4],{2,4}]]];
```

```
The clustering coeff. of the blue node in the second graph of fig. 9 is: 2/3
```

The *clustering coefficient for the entire network* is the average of the clustering coefficients of all the nodes. Some analysts use a "weighted" version of the clustering coefficient, giving a weight to the neighbourhood densities proportional to their size (i.e. actors with larger neighbourhoods get more weight in computing the average density).

```
myListOfClusteringCoefficients=Clustering[RandomGraph[100,0.6]];
minDegree=Min[myListOfClusteringCoefficients[[All,1]]];
ListPlot[myListOfClusteringCoefficients,
        AxesOrigin->{minDegree,0}, PlotRange->{{minDegree,Automatic},{0,1}},
        FrameLabel->{"# of Neighbours","Avg. Clustering Coefficient"}, Frame->True];
```




```
Print["The clustering coefficient list of my fragmented grid is: ",
      Clustering[myFragmentedGrid]];
```

```
The clustering coefficient list of my fragmented grid is:  {{2,0},{3,0},{4,0}}
```

8 CENTRALITY AND POWER

The *centrality* of a node in a network is a measure of its structural importance (for example, how important a person is within a social network, how important a room is within a building, or how important a road is within an urban network). Given the subjectivity of the term "importance", it is not surprising that there are various measures of centrality in Graph Theory. All of them aim at quantifying the prominence of an individual actor embedded in a network, but they differ on the criteria used to achieve that.

There are three approaches to calculate the centrality of a node: based on *degree*, on *closeness*, and on *betweenness*. *Degree* approaches are based on the idea that having more ties means being more important. *Closeness* approaches go slightly further and they assume that actors who are able to reach other actors at shorter path lengths, or who are more reachable by others at shorter path lengths, are in favoured positions. Finally, when using *betweenness* approaches, it is being in between many other actors what makes an actor central.

These three approaches describe the locations of nodes in terms of how close they are to the "centre" of the action in a network - though the definitions of what it means to be at the centre differ. A central actor, presumably, has a stronger influence on other network members (i.e. central positions tend to be powerful positions). Thus, measures of centrality are often interpreted as measures of power. However, because of the vague meaning of the word "power", network analysts tend to describe their approaches as measures of centrality rather than of power.

8.1 Degree centrality

8.1.1 Simple degree centrality

Actors who have many ties with other actors may be in an advantageous position. Having many ties may mean having alternative ways of satisfying needs, it may mean having access to more resources, and it may also mean acting frequently as a third-party or deal maker in exchanges among others. So, a very simple, but often very effective, measure of an actor's centrality is its degree.

In undirected data, actors differ from one another only in how many connections they have. With directed data, however, it is important to distinguish between *in-degree centrality* and *out-degree centrality*. If an actor receives many ties, they are often said to be *prominent*, or to have *high prestige*. That is, many other actors seek to direct ties towards them, and this may be an indicator of importance. Actors with unusually high out-degree may be able to influence many others, or make many others aware of their views. Thus, actors with high out-degree centrality are often called *influential* actors.

8.1.2 Bonacich's approach to degree centrality

Bonacich (1987) argued that an actor's centrality is a function of how many connections the actor has, but also on how many connections the actor's social neighbours have.

While we have argued that more central actors tend to be more powerful actors, Bonacich questioned this idea. If the actors that you are connected to are, themselves, well connected, they are not highly dependent on you. If, on the other hand, the people to whom you are connected are not, themselves, well connected, then they are dependent on you. Bonacich argued that being connected to well-connected others makes an actor central, but not powerful. Somewhat ironically, being connected to others that are not well connected makes one powerful, because these other actors are dependent on you.

Bonacich's degree centrality index has a parameter called "attenuation factor" that determines the effect of an actor's neighbour's connections on the actor's power. Calculating this index often requires an iterative approach. For more information, see http://faculty.ucr.edu/~hanneman/nettext/C10_Centrality.html#Bonacich , or Bonacich (1987).

8.2 Closeness centrality

Degree centrality measures might be criticised because they only take into account the immediate ties that an actor has (and the ties of the actor's neighbours when using Bonacich's approach), rather than indirect ties to all others. Closeness approaches aim to amend this by considering the distance from each actor to all others.

8.2.1 Closeness centrality using geodesic distance

Closeness centrality using geodesic distance is the reciprocal of the sum of geodesic distances to all other vertices in the graph. These scores can be normalised dividing by the maximum value.

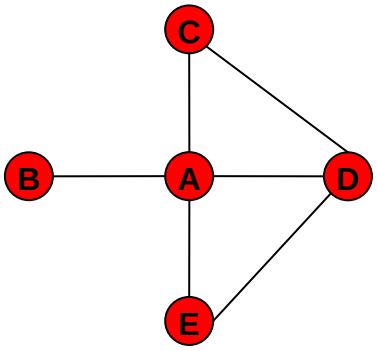


Figure 10. An undirected graph. Node A is the most central node, whilst node B is the least central.

Closeness Centralities using geodesic distance:

$$\text{Node A} = \frac{1}{1+1+1+1} = 0.25$$

$$\text{Node B} = \frac{1}{1+2+2+2} = 0.14$$

$$\text{Node C} = \frac{1}{1+2+1+2} = 0.17$$

$$\text{Node D} = \frac{1}{1+2+1+1} = 0.2$$

$$\text{Node E} = \frac{1}{1+2+2+1} = 0.17$$

```
Needs["DiscreteMath`Combinatorica`"];
graphFig10=AddEdges[EmptyGraph[5],{{1,2},{1,3},{1,4},{1,5},{3,4},{4,5}}];
N[GeodesicCloseness[graphFig10]]
```

```
{0.25, 0.142857, 0.166667, 0.2, 0.166667}
```

8.2.2 Closeness centrality using reachability

Another way of thinking about how close an actor A is to all others is to calculate the proportion of other actors that A can reach in one step, two steps, three steps, etc (or, alternatively, the proportion of nodes that reach A in *n* steps). One can then calculate a single index for each node by summing up the proportion of other nodes reached (for the first time) at a given distance, appropriately weighted (e.g. 1 for nodes at distance 1, ½ for nodes at distance 2…). These scores can be then normalised dividing by the maximum value, if this is considered appropriate.

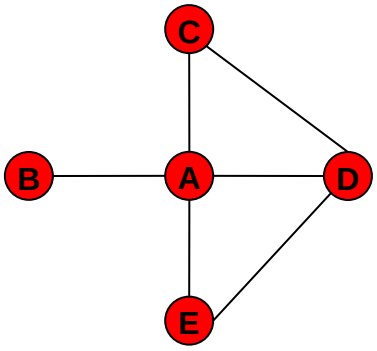


Figure 11. An undirected graph. Node A is the most central node, whilst node B is the least central.

Closeness Centralities using reachability:

$$\text{Node A} = 1 \cdot \frac{4}{4} = 1$$

$$\text{Node B} = 1 \cdot \frac{1}{4} + \frac{1}{2} \cdot \frac{3}{4} = 0.625$$

$$\text{Node C} = 1 \cdot \frac{2}{4} + \frac{1}{2} \cdot \frac{2}{4} = 0.75$$

$$\text{Node D} = 1 \cdot \frac{3}{4} + \frac{1}{2} \cdot \frac{1}{4} = 0.875$$

$$\text{Node E} = 1 \cdot \frac{2}{4} + \frac{1}{2} \cdot \frac{2}{4} = 0.75$$

```
Needs["Statistics`DataManipulation`"];
N[ReachabilityCloseness[graphFig10]]
```

```
{1., 0.625, 0.75, 0.875, 0.75}
```

8.3 Betweenness centrality

The idea behind betweenness centrality is that being in between actors makes you powerful because you may be able to control the flow of e.g. information, resources, gossip… between them. Nodes with high betweenness are often called key-players.

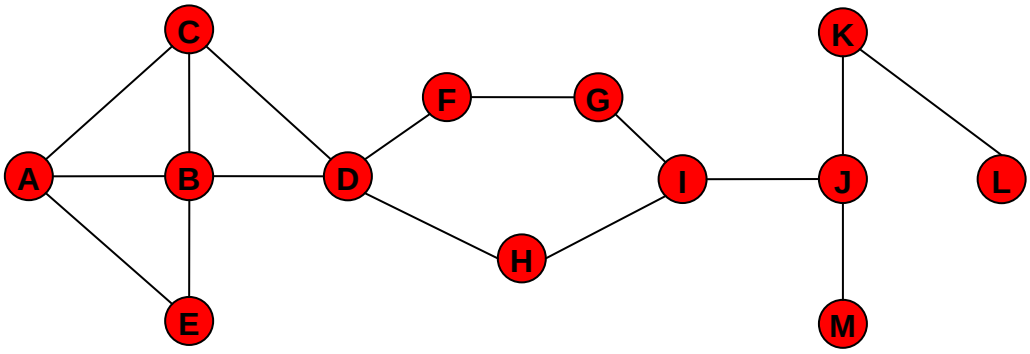


Figure 12. An undirected network. Nodes D and I have higher betweenness than the rest of the nodes.

8.3.1 Betweenness centrality using only shortest paths

Under this approach, nodes that occur on many shortest paths between other nodes have higher betweenness than those that do not. The betweenness of a node A is the fraction of all the possible shortest paths between any two nodes in the network (excluding A) that pass through A.

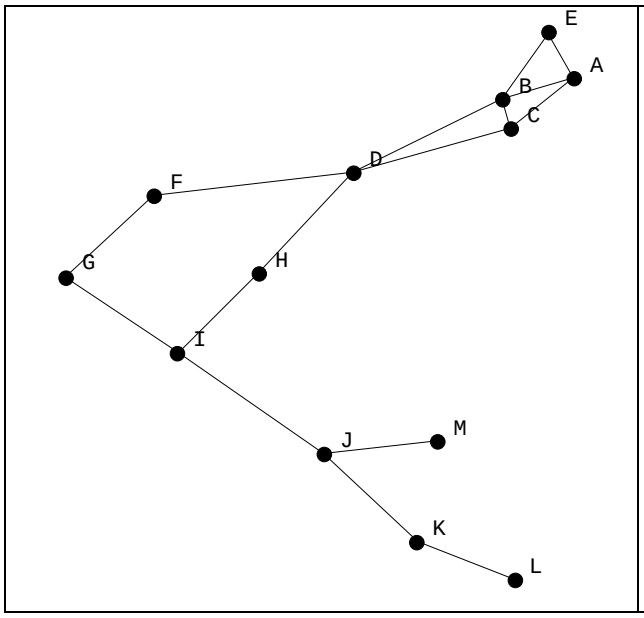
Formally, for a graph $G := (V, E)$ with n vertices (with edges), the betweenness centrality using shortest paths $C_{B\text{-}SP}(v)$ for vertex v is:

$$C_{B-SP}(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} is the number of shortest geodesic paths from s to t, and $\sigma_{st}(v)$ the number of shortest geodesic paths from s to t that pass through vertex v. This may be normalised by dividing by the number of pairs of vertices not including v, which is $(n - 1)\cdot(n - 2)$.

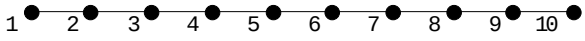
The betweenness centrality of each node in the network shown in Figure 12 using only shortest paths is:

```
N[Inner[List, vertexLabels, Betweenness[graphFig12], List]] // MatrixForm
```



A	0.0149254
B	0.25
C	0.12
D	0.546667
E	0.
F	0.08
G	0.0666667
H	0.4
I	0.493333
J	0.426667
K	0.16
L	0.
M	0.

```
g=Path[10];
ShowGraph[g,VertexNumber→True];
Transpose[Inner[List,Range[10],N[Betweenness[g]],List]]//MatrixForm
```



1	2	3	4	5	6	7	8	9	10
0.	0.222222	0.388889	0.5	0.555556	0.555556	0.5	0.388889	0.222222	0.

8.3.2 Betweenness centrality using flow

The use of shortest paths to analyse betweenness centrality is often very useful. However, there may be other cases where considering all connections among actors – not just the most efficient ones – may be more appropriate. Rumours, for instance, may spread in a network through all pathways – not just the most efficient ones. Similarly, how much credibility a person gives to a rumour may depend on how many times they hear it from different sources – not on how soon they hear it.

Thus, there are many networks where paths that are not necessarily the shortest will do the job (e.g. of transmitting information, resources…) almost as well as shortest paths, particularly if the latter are problematic for some reason. The flow approach to centrality expands the notion of betweenness centrality in this way. It assumes that actors will use all pathways that connect them; the importance of each path can be weighted according to its length.

Formally, for a graph G: = (V, E) with n vertices (with edges), the betweenness centrality using flow $C_{B\text{-}F}(v)$ for vertex v is:

$$C_{B-F}(v) = \sum_{i=2}^{n} \left(w_i \cdot \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(i,v)}{\sigma_{st}(i)} \right)$$

where w_i are the weights assigned to paths of length i, $\sigma_{st}(i)$ is the number of paths of length i from s to t, and $\sigma_{st}(i, v)$ the number of paths of length i from s to t that pass through vertex v. This may be normalised by dividing by the number of pairs of vertices not including v, which is $(n - 1)\cdot(n - 2)$.

9 REFERENCES

Bonacich, P. (1987). Power and centrality: a family of measures. *American Journal of Sociology* 92, 1170-1182.

Hanneman, R. A. and M. Riddle (2005). *Introduction to Social Network Methods*. Riverside, CA: University of California, Riverside (published in digital form at http://www.faculty.ucr.edu/~hanneman/nettext/)

Janssen, M. A., Ö. Bodin, J. M. Anderies, T. Elmqvist, H. Ernstson, R. R. J. McAllister, P. Olsson, and P. Ryan (2006). A network perspective on the resilience of social-ecological systems. *Ecology and Society* 11(1): 15. http://www.ecologyandsociety.org/vol11/iss1/art15/.

Watts, D.J. (1999). *Small Worlds: The Dynamics of Networks between Order and Randomness*. Princeton University Press, Princeton, NJ.

Introduction to the formal analysis of social networks

[redacted]

Packages

```
Needs["DiscreteMath`Combinatorica`"]; (* For the analysis of graphs. *)
Needs["Graphics`Graphics`"]; (* For the Histograms. *)
Needs["Statistics`DataManipulation`"];
(* For Frequencies, used in the function ReachabilityCloseness. *)
```

Functions

```
(* This function returns the number
  of reciprocated links in a simple network.*)

ReciprocatedEdges[g_Graph] := Module[{adjMatrix},
    adjMatrix = ToAdjacencyMatrix[g];
    Count[adjMatrix + Transpose[adjMatrix], 2, {2}]
   ];


 (* A clique is a subset of the vertices such that every
   pair of vertices in the subset is connected by an edge. This
   function returns a clique of maximum size allowing for paths
   of any length no greater than n (i.e. an n-clique). *)

 MaximumNClique[g_Graph, n_Integer] :=
   MaximumClique[RemoveSelfLoops[RemoveMultipleEdges[GraphPower[g, n]]]];


 (* Returns true if c is an n-
   clique (i.e. allowing for paths of any length no greater than n).
     False otherwise. *)

 NCliqueQ[g_Graph, n_Integer, c_List] :=
   CliqueQ[RemoveSelfLoops[RemoveMultipleEdges[GraphPower[g, n]]], c];


 (* An n-clan is an n-clique where all ties among members of the n-
   clique occur through members of the n-
   clique. This function returns a list of all the n-clans of maximum
    size (i.e. allowing for paths of any length no greater than n). *)

 MaximumNClans[g_Graph, n_Integer] :=
   Module[{listOfDegrees, allSuitableNodes = Range[V[g]], clans = {}},

    (* A clan is, in particular, a clique,
     so find out the maximum possible size of the clan. *)
    maxSizeOfClan = Length[MaximumNClique[g, n]];

    (* There is no need to consider the nodes of degree maxSizeOfClan-
      2 or less (in the powered graph) as they cannot be part of an n-
```

```
    clique or an n-clan. *)
  listOfDegrees = Degrees[RemoveSelfLoops[
     RemoveMultipleEdges[GraphPower[g, n]]]];
  allSuitableNodes = Select[allSuitableNodes,
    (listOfDegrees[[#]] ≥ maxSizeOfClan - 1) &];

  While[Length[clans] == 0 && maxSizeOfClan ≥ 1,
   possibleClans = Subsets[allSuitableNodes, {maxSizeOfClan}];
   clans = Select[possibleClans, CompleteQ[RemoveSelfLoops[
        RemoveMultipleEdges[GraphPower[InduceSubgraph[g, #], n]]]] &];
   maxSizeOfClan--;
  ];
  clans
 ];


(* Returns true if c is an n-
  clan (i.e. allowing for paths of any length no greater than n). False
   otherwise. *)

NClanQ[g_Graph, n_Integer, c_List] := CompleteQ[RemoveSelfLoops[
    RemoveMultipleEdges[GraphPower[InduceSubgraph[g, c], n]]]];


(* The clustering coefficient of a node is the ratio of
  existing links connecting the node's neighbours to each other,
 to the maximum possible number of such links. For nodes with fewer
  than two neighbours the clustering coefficient is undefined.*)

Clustering[v_Integer, g_Graph] := Module[
   {nbrs, subGraph},

   nbrs = Complement[Neighborhood[g, v, 1], {v}];
   If[Length[nbrs] < 2, Return["Undefined"]];

   subGraph = InduceSubgraph[g, nbrs];

   If[UndirectedQ[g],
    2 * M[subGraph] / (V[subGraph] * (V[subGraph] - 1)),
    M[subGraph] / (V[subGraph] * (V[subGraph] - 1))
   ]
  ];


(* This function returns a list of 2-
  dimensional vectors. The first component of each vector
   is a number nbrs denoting a number of neighbours. The
```

```
    second component is the average number of edges between
    the neighbours of every node that has nbrs neighbours. It
    works for both directed and undirected graphs. *)

Clustering[g_Graph] := Module[
    {listOfNeighbourhoods,
     listOfNeighbourhoodsGreaterThan1, listOfSubgraphs,
     myClusteringList, nOfNbrs, listOfDiffOutDegress, totalNumEdges},

    listOfNeighbourhoods =
     Map[Complement[Neighborhood[g, #, 1], {#}] &, Range[V[g]]];
    listOfNeighbourhoodsGreaterThan1 =
     Select[listOfNeighbourhoods, (Length[#] > 1) &];
    listOfSubgraphs = Map[InduceSubgraph[g, #] &,
      listOfNeighbourhoodsGreaterThan1];

    (* myClusteringList is a list of two
      elements for each node with 2 or more neighbours. The
      first element of the list is the number of neighbours,
     and the second element is the number of edges between the
      neighbours. It works for both directed and undirected graphs. *)
    myClusteringList = Map[{V[#], M[#]} &, listOfSubgraphs];

    nOfNbrs = Length[listOfNeighbourhoodsGreaterThan1];
    listOfDiffOutDegress = Union[myClusteringList[[All, 1]]];
    totalNumEdges = Table[0, {Length[listOfDiffOutDegress]}];

    For[i = 1, i ≤ nOfNbrs, i++,
     totalNumEdges[[Position[listOfDiffOutDegress, myClusteringList[[i,
             1]], 1, 1][[1, 1]]]] += myClusteringList[[i, 2]];
    ];

    If[UndirectedQ[g],
     Inner[List, listOfDiffOutDegress,
      2 * totalNumEdges / (listOfDiffOutDegress * (listOfDiffOutDegress - 1) *
          Map[Count[myClusteringList[[All, 1]], #] &,
           listOfDiffOutDegress]), List],
     Inner[List, listOfDiffOutDegress, totalNumEdges /
       (listOfDiffOutDegress * (listOfDiffOutDegress - 1) *
        Map[Count[myClusteringList[[All, 1]], #] &,
         listOfDiffOutDegress]), List],
    ]
   ];


 (* The closeness centrality of a node using geodesic distance is the
   reciprocal of the sum of geodesic distances from the node to all
```

```
  other vertices in the graph. This function returns the closeness
  centrality using geodesic distance of every node in the graph. *)

GeodesicCloseness[g_Graph] := 1 / Total[AllPairsShortestPath[g]];


(* This function returns a list with the reachability closeness
  of each node in the graph. The reachability closeness of a
  node is calculated by summing up the proportion of other
  nodes reached (for the first time) at a given distance,
 appropriately weighted. In this case the weights used are
  1/n where n denotes the distance between the nodes
  (e.g. 1 for nodes at distance 1, ½ for nodes at distance 2…) *)

ReachabilityCloseness[g_Graph] := Module[{freqLists},
  freqLists = Map[Rest[Frequencies[#]] &, AllPairsShortestPath[g]];
  Map[(Total[#[[All, 1]] * (1 / #[[All, 2]])]) &, freqLists] / (V[g] - 1)
 ]


(* This function returns a matrix indicating the number
  of shortest paths between any two nodes in the network.*)

Options[NumberOfShortestPaths] = {CheckGraphIsConnected → True};
NumberOfShortestPaths[g_Graph, opts___?OptionQ] := Module[
  {distancesMatrix, adjMatrix, powersOfAdjMatrix},

  checkForConnected = CheckGraphIsConnected /.
    Flatten[{opts, Options[NumberOfShortestPaths]}];
  (* Replace (/.) returns the result from using
    the first rule that applies. *)
  If[checkForConnected,
   If[(UndirectedQ[g] && !ConnectedQ[g]) ||
      (!UndirectedQ[g] && !ConnectedQ[g, Strong]),
     Print["The graph is not (strongly) connected"]; Return[];
    ];
  ];

  distancesMatrix = AllPairsShortestPath[g];
  If[Not[checkForConnected],
   distancesMatrix = Replace[distancesMatrix, ∞ → 0, {2}]];
  (* A distance of 0 will give 0 shortest paths between those nodes. *)

  adjMatrix = ToAdjacencyMatrix[g];
  powersOfAdjMatrix =
   NestList[(#.adjMatrix) &, adjMatrix, Max[distancesMatrix] - 1];
```

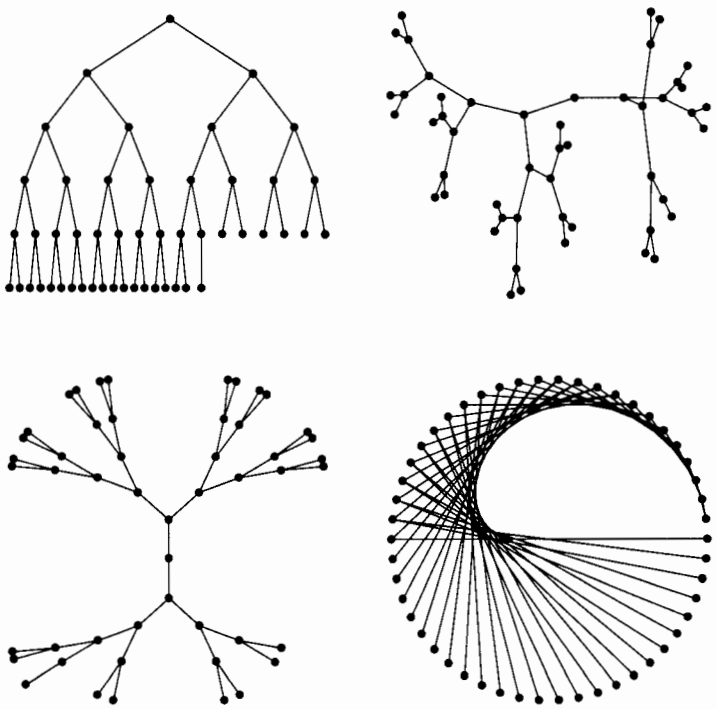
```
  PrependTo[powersOfAdjMatrix, Table[0, {V[g]}, {V[g]}]];
  (* We prepend the matrix that pairs showing a distance of 0
    will read. These pairs are those that are not connected,
   and the pairs consisting of a node with itself. *)
  MapIndexed[(powersOfAdjMatrix[[#1, Apply[Sequence, #2]]]) &,
   distancesMatrix + Table[1, {V[g]}, {V[g]}], {2}]
 ]


(* This function returns the betweeness of node v. The betweenness
  of a node v is the fraction of all the possible shortest
  paths between any two nodes in the network (excluding v) that
  pass through v. The idea in this algorithm is that the number
  of shortest paths between i and j that do not pass through
  a node k are the number of paths between i and j OF LENGTH
  EQUAL TO THE DISTANCE BETWEEN i AND j IN THE ORIGINAL GRAPH,
 in a graph where node k has been deleted. *)

Betweenness[v_Integer, g_Graph] := Module[{distancesMatrix, adjMatrix,
    powersOfAdjMatrix, nShortestPaths, adjMatrixWithoutVertex,
    powersOfAdjMatrixWithoutVertex, nShortestPathsNotThroughV},

   If[(UndirectedQ[g] && ! ConnectedQ[g]) ||
     (! UndirectedQ[g] && ! ConnectedQ[g, Strong]),
    Print["The graph is not (strongly) connected"]; Return[];
   ];

   distancesMatrix = AllPairsShortestPath[g];

   (* Calculate the number of
     shortest paths between any two nodes in the graph *)
   adjMatrix = ToAdjacencyMatrix[g];
   powersOfAdjMatrix =
    NestList[(#.adjMatrix) &, adjMatrix, Max[distancesMatrix] - 1];
   PrependTo[powersOfAdjMatrix, Table[0, {V[g]}, {V[g]}]];
   (* We prepend the matrix that pairs showing a distance of 0
     will read. These pairs are those that are not connected,
    and the pairs consisting of a node with itself. *)
   nShortestPaths = MapIndexed[
     (powersOfAdjMatrix[[#1, Apply[Sequence, #2]]]) &,
     distancesMatrix + Table[1, {V[g]}, {V[g]}], {2}];

   (* Calculate the number of paths between i and j OF LENGTH EQUAL
     TO THE DISTANCE BETWEEN i AND j IN THE ORIGINAL GRAPH,
    in a graph where node v has been deleted. This is the
     number of shortest paths between i and j that do
     not pass through node v in the original graph. *)
```

```
    adjMatrixWithoutVertex = Drop[adjMatrix, {v}, {v}];
    powersOfAdjMatrixWithoutVertex =
     NestList[(#.adjMatrixWithoutVertex) &,
      adjMatrixWithoutVertex, Max[distancesMatrix] - 1];
    PrependTo[powersOfAdjMatrixWithoutVertex, Table[0, {V[g]}, {V[g]}]];
    (* We prepend the matrix that pairs showing a distance of 0
      will read. These pairs are those that are not connected,
     and the pairs consisting of a node with itself. *)
    nShortestPathsNotThroughV = MapIndexed[
      (powersOfAdjMatrixWithoutVertex[[#1, Apply[Sequence, #2]]]) &,
      Drop[distancesMatrix, {v}, {v}] + Table[1, {V[g] - 1}, {V[g] - 1}], {2}];

    (* Return the betweenness *)
    (1 - Total[nShortestPathsNotThroughV, 2] /
       Total[Drop[nShortestPaths, {v}, {v}], 2])
  ];


(* This function returns a list with the betweeness of every node in the
  network. The betweenness of a node v is the fraction of all the
  possible shortest paths between any two nodes in the network
  (excluding v) that pass through v. The idea in this algorithm is
  that the number of shortest paths between i and j that do not
  pass through a node k are the number of paths between i and j OF
  LENGTH EQUAL TO THE DISTANCE BETWEEN i AND j IN THE ORIGINAL GRAPH,
 in a graph where node k has been deleted. *)

Betweenness[g_Graph] := Module[
    {nNodes, distancesMatrix, adjMatrix, powersOfAdjMatrix, nShortestPaths,
     v, adjMatrixWithoutVertex, powersOfAdjMatrixWithoutVertex,
     nShortestPathsNotThroughV, betweennessList},

    If[(UndirectedQ[g] && ! ConnectedQ[g]) ||
      (! UndirectedQ[g] && ! ConnectedQ[g, Strong]),
     Print["The graph is not (strongly) connected"]; Return[];
    ];

    distancesMatrix = AllPairsShortestPath[g];
    nNodes = V[g];

    (* Calculate the number of
      shortest paths between any two nodes in the graph *)
    adjMatrix = ToAdjacencyMatrix[g];
    powersOfAdjMatrix =
     NestList[(#.adjMatrix) &, adjMatrix, Max[distancesMatrix] - 1];
    PrependTo[powersOfAdjMatrix, Table[0, {nNodes}, {nNodes}]];
    (* We prepend the matrix that pairs showing a distance of 0
```

```
  will read. These pairs are those that are not connected,
 and the pairs consisting of a node with itself. *)
nShortestPaths = MapIndexed[
   (powersOfAdjMatrix[[#1, Apply[Sequence, #2]]]) &,
   distancesMatrix + Table[1, {nNodes}, {nNodes}], {2}];

betweennessList = {};
(* Calculate the number of paths between i and j OF LENGTH EQUAL
  TO THE DISTANCE BETWEEN i AND j IN THE ORIGINAL GRAPH,
 in a graph where node v has been deleted. This is the
  number of shortest paths between i and j that do
  not pass through node v in the original graph. *)
For[v = 1, v ≤ nNodes, v++,
 adjMatrixWithoutVertex = Drop[adjMatrix, {v}, {v}];
 powersOfAdjMatrixWithoutVertex =
  NestList[(#.adjMatrixWithoutVertex) &,
   adjMatrixWithoutVertex, Max[distancesMatrix] - 1];
 PrependTo[powersOfAdjMatrixWithoutVertex, Table[0, {nNodes},
   {nNodes}]]; (* We prepend the matrix that pairs showing a distance
   of 0 will read. These pairs are those that are not connected,
  and the pairs consisting of a node with itself. *)
 nShortestPathsNotThroughV = MapIndexed[
   (powersOfAdjMatrixWithoutVertex[[#1, Apply[Sequence, #2]]]) &,
   Drop[distancesMatrix, {v}, {v}] +
    Table[1, {nNodes - 1}, {nNodes - 1}], {2}];

 AppendTo[betweennessList, 1 - Total[nShortestPathsNotThroughV, 2] /
    Total[Drop[nShortestPaths, {v}, {v}], 2]];
];

(* Return the betweenness *)
betweennessList
];
```

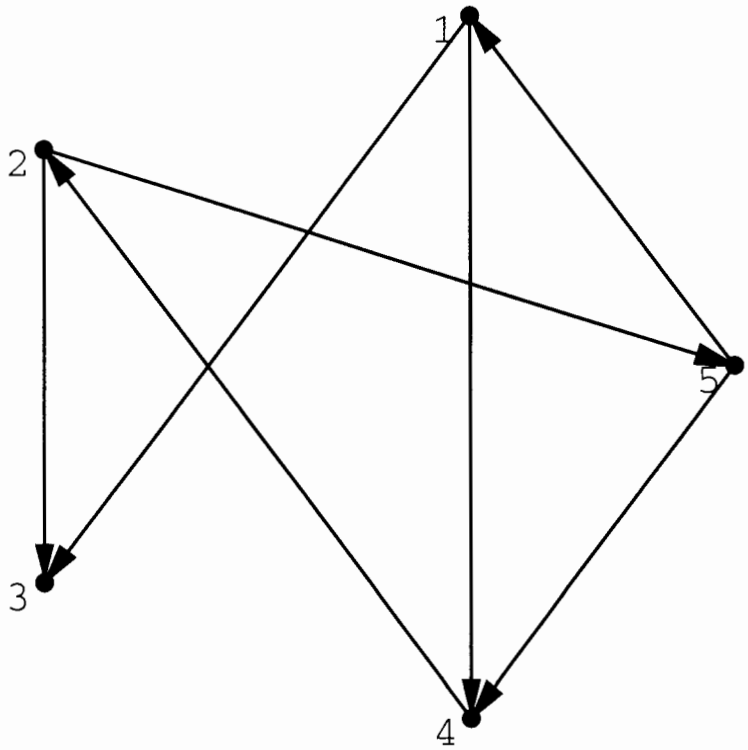
Basics Concepts and Terminology

```
myBinaryTree = CompleteBinaryTree[50];

g1 = ShowGraph[myBinaryTree, DisplayFunction → Identity];
g2 = ShowGraph[
    SpringEmbedding[myBinaryTree, 200, 0.05], DisplayFunction → Identity];
g3 = ShowGraph[RadialEmbedding[myBinaryTree], DisplayFunction → Identity];
g4 =
  ShowGraph[CircularEmbedding[myBinaryTree], DisplayFunction → Identity];

Show[GraphicsArray[{{g1, g2}, {g3, g4}}]];
```



Representing Social Network Data

```
SeedRandom[2];

myGraph = RandomGraph[5, 0.3, Type → Directed];
ShowGraph[myGraph, VertexNumber → True,
  TextStyle → {FontSize → 16}, VertexNumberPosition → {-0.01, -0.02}];
```



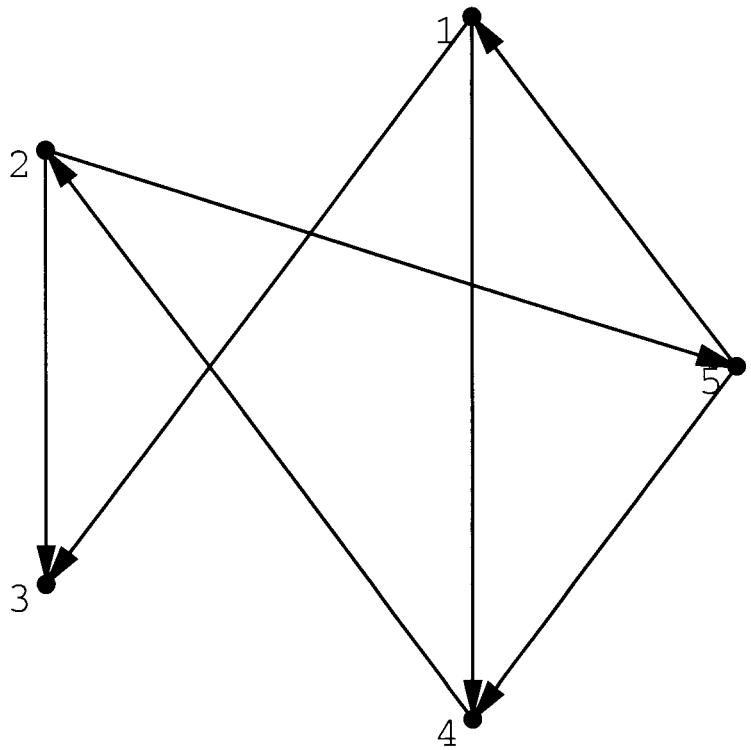
```
myAdjMatrix = ToAdjacencyMatrix[myGraph];
Table[MatrixForm[MatrixPower[myAdjMatrix, i]], {i, 5}]
```

$$\left\{\begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix},\right.$$

$$\left.\begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 2 & 0 \end{pmatrix}\right\}$$

Basic Properties of Networks and Actors

```
SeedRandom[2];

myGraph = RandomGraph[5, 0.3, Type → Directed];
ShowGraph[myGraph, VertexNumber → True,
  TextStyle → {FontSize → 16}, VertexNumberPosition → {-0.01, -0.02}];
```




```
Print["Number of nodes: ", V[myGraph]];
Print["Number of edges: ", M[myGraph]];
```

```
Number of nodes: 5

Number of edges: 7
```

```
If[UndirectedQ[myGraph],
  Print["Density: ", 2 M[myGraph] / (V[myGraph] (V[myGraph] - 1))],
  Print["Density: ", M[myGraph] / (V[myGraph] (V[myGraph] - 1))]];
```

Density: $\frac{7}{20}$

```
Print["The in-degree of each node in the network is: ",
  InDegree[myGraph]];
Print["The out-degree of each node in the network is: ",
  OutDegree[myGraph]];
```

The in-degree of each node in the network is: {1, 1, 2, 2, 1}

The out-degree of each node in the network is: {2, 2, 0, 1, 2}

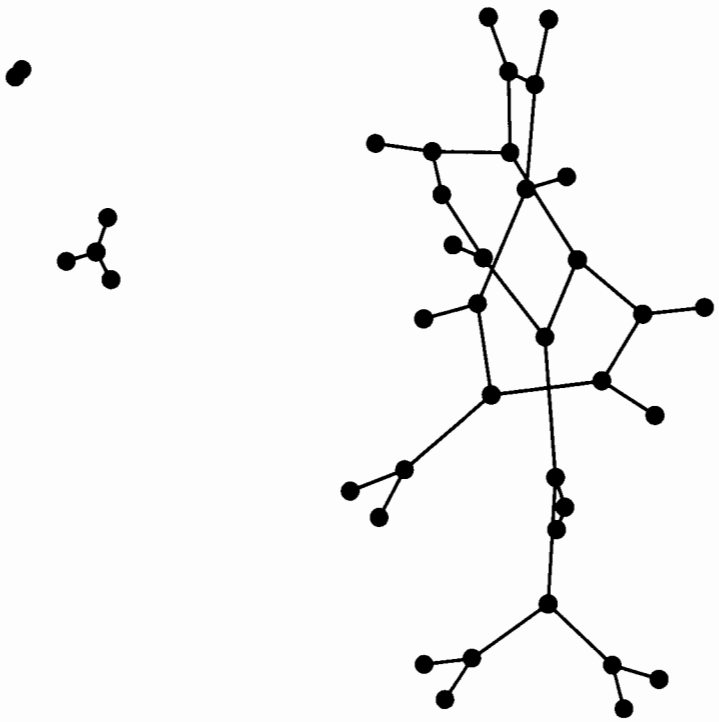
```
DegreeSequence[MakeUndirected[myGraph]]
```

```
{3, 3, 3, 3, 2}
```

```
myList = DegreeSequence[CompleteBinaryTree[40]]
```

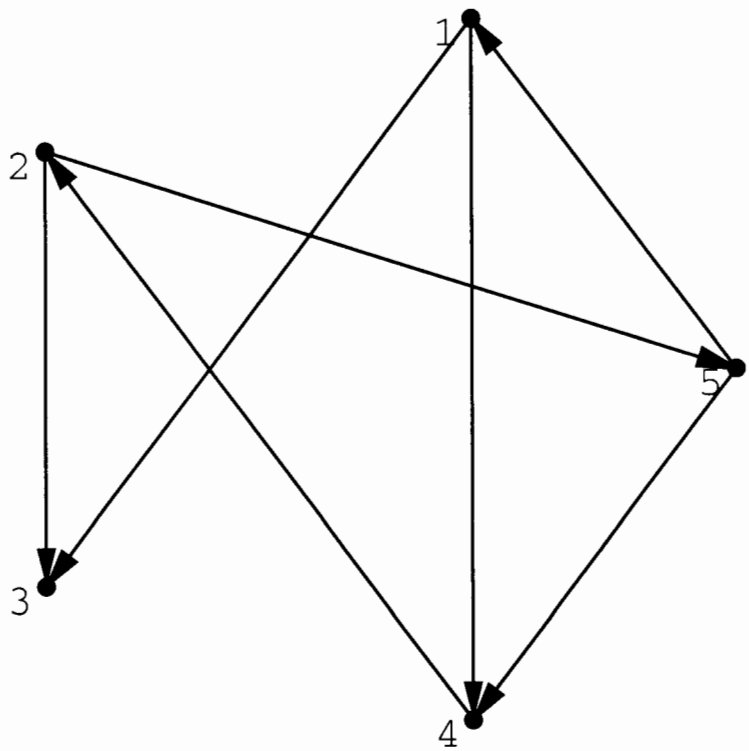
```
{3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2,
 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
```

```
ShowGraph[SpringEmbedding[RealizeDegreeSequence[myList], 100, 0.05]];
```



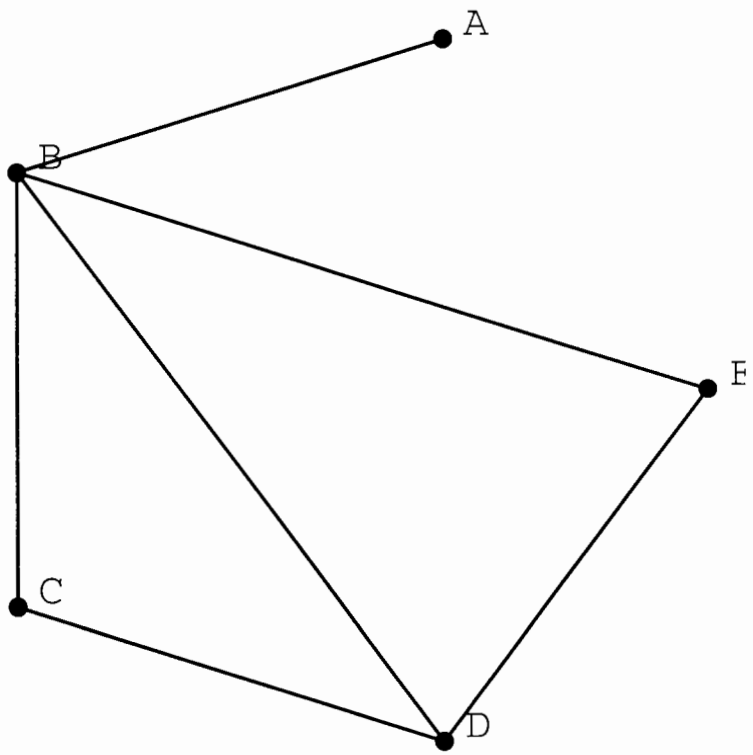
Social Distance and Related Concepts

```
SeedRandom[2];

myGraph = RandomGraph[5, 0.3, Type → Directed];
ShowGraph[myGraph, VertexNumber → True,
  TextStyle → {FontSize → 16}, VertexNumberPosition → {-0.01, -0.02}];

Print["All pairs distances: ", MatrixForm[AllPairsShortestPath[myGraph]]];
```



All pairs distances:
$$\begin{pmatrix} 0 & 2 & 1 & 1 & 3 \\ 2 & 0 & 1 & 2 & 1 \\ \infty & \infty & 0 & \infty & \infty \\ 3 & 1 & 2 & 0 & 2 \\ 1 & 2 & 2 & 1 & 0 \end{pmatrix}$$

```
myAdjMatrix = {{0, 1, 0, 0, 0}, {1, 0, 1, 1, 1},
   {0, 1, 0, 1, 0}, {0, 1, 1, 0, 1}, {0, 1, 0, 1, 0}};
graphFig6 = SetVertexLabels[FromAdjacencyMatrix[myAdjMatrix],
   {A, B, C, D, "E"}];
ShowGraph[graphFig6, TextStyle → {FontSize → 16}];
```



```
Table[Print["Number of walks of length ", i, " between nodes: ",
   MatrixForm[MatrixPower[myAdjMatrix, i]]], {i, 4}];
```

Number of walks of length 1 between nodes:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Number of walks of length 2 between nodes:

$$\begin{pmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 4 & 1 & 2 & 1 \\ 1 & 1 & 2 & 1 & 2 \\ 1 & 2 & 1 & 3 & 1 \\ 1 & 1 & 2 & 1 & 2 \end{pmatrix}$$

Number of walks of length 3 between nodes:

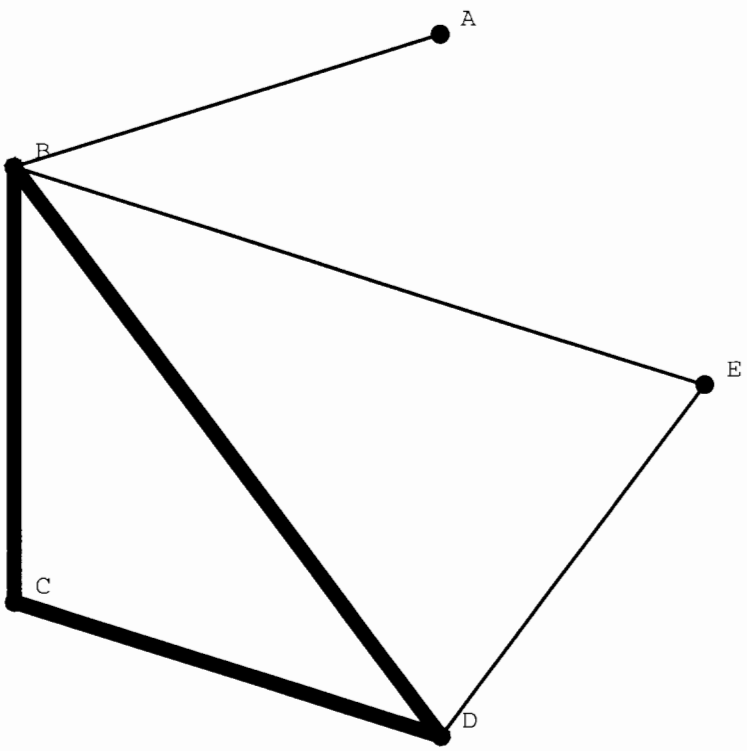
$$\begin{pmatrix} 0 & 4 & 1 & 2 & 1 \\ 4 & 4 & 6 & 6 & 6 \\ 1 & 6 & 2 & 5 & 2 \\ 2 & 6 & 5 & 4 & 5 \\ 1 & 6 & 2 & 5 & 2 \end{pmatrix}$$

Number of walks of length 4 between nodes:

$$\begin{pmatrix} 4 & 4 & 6 & 6 & 6 \\ 4 & 22 & 10 & 16 & 10 \\ 6 & 10 & 11 & 10 & 11 \\ 6 & 16 & 10 & 16 & 10 \\ 6 & 10 & 11 & 10 & 11 \end{pmatrix}$$

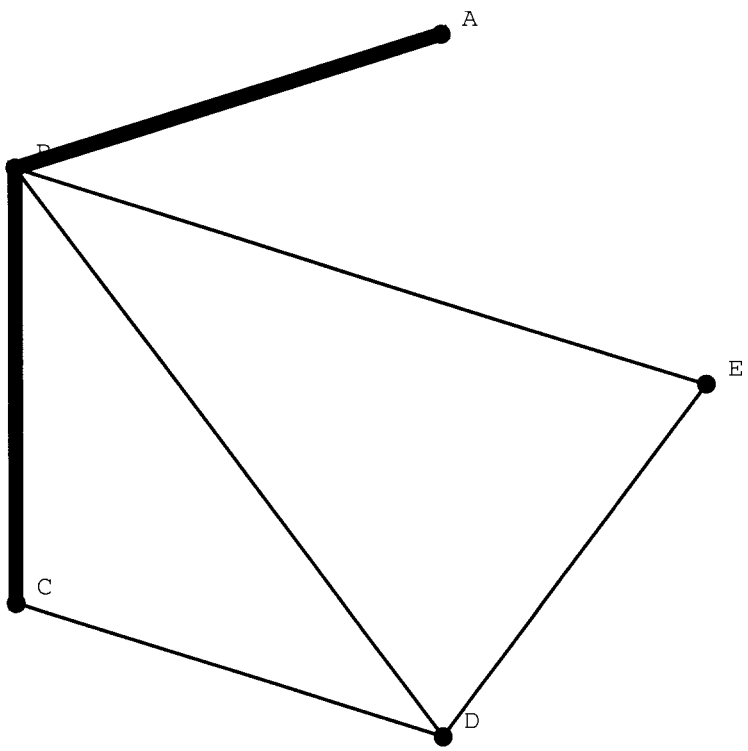
```
? ExtractCycles
ShowGraph[
  Highlight[graphFig6, {Partition[ExtractCycles[graphFig6][[1]], 2, 1]}]];
```

```
ExtractCycles[g] gives a maximal list of edge-disjoint cycles in graph g.
```
More...



```
Print["Shortest path between A (node 1) and C (node 3): ",
  ShortestPath[graphFig6, 1, 3]];
ShowGraph[Highlight[graphFig6,
   {Partition[ShortestPath[graphFig6, 1, 3], 2, 1]}]];
```

```
Shortest path between A (node 1) and C (node 3): {1, 2, 3}
```



```
Print["The eccentricity of each node in the network is: ",
  Eccentricity[graphFig6]];
Print["The diameter of the network is: ", Diameter[graphFig6]];
Print["The radius of the network is: ", Radius[graphFig6]];
Print["The peripheral vertices are: ",
  Position[Eccentricity[graphFig6], Diameter[graphFig6]]];
Print["The nodes in the centre of the network are: ",
  GraphCenter[graphFig6]];
```

```
The eccentricity of each node in the network is: {2, 1, 2, 2, 2}
The diameter of the network is: 2
The radius of the network is: 1
The peripheral vertices are: {{1}, {3}, {4}, {5}}
The nodes in the centre of the network are: {2}
```

Connection and Connectivity

```
Print["Is the graph in fig. 6 connected? ", ConnectedQ[graphFig6]];
```

```
Is the graph in fig. 6 connected? True
```

```
Print["Is my graph strongly connected? ", ConnectedQ[myGraph, Strong]];
Print["The strongly connected components in my graph are: ",
  StronglyConnectedComponents[myGraph]];
```

```
Is my graph strongly connected? False
The strongly connected components in my graph are: {{1, 2, 4, 5}, {3}}
```

```
Print["Is my graph weakly connected? ", ConnectedQ[myGraph, Weak]];
Print["The weakly connected components in my graph are: ",
  WeaklyConnectedComponents[myGraph]];
```

```
Is my graph weakly connected? True
The weakly connected components in my graph are: {{1, 2, 3, 4, 5}}
```

```
? VertexConnectivity
Print["The vertex-connectivity of graph in fig. 6 is: ",
  VertexConnectivity[graphFig6]];
```

```
VertexConnectivity[g] gives the minimum number of vertices whose deletion
   from graph g disconnects it. VertexConnectivity[g, Cut] gives a set
   of vertices of minimum size, whose removal disconnects the graph. More...
```

```
The vertex-connectivity of graph in fig. 6 is: 1
```

```
? ArticulationVertices
Print["The articulation vertices of graph in fig. 6 are: ",
  ArticulationVertices[graphFig6]];
```

```
ArticulationVertices[g] gives a list of all articulation vertices in graph
   g. These are vertices whose removal will disconnect the graph. More...
```

```
The articulation vertices of graph in fig. 6 are: {2}
```

```
? EdgeConnectivity
Print["The edge-connectivity of graph in fig. 6 is: ",
  EdgeConnectivity[graphFig6]];
```

```
EdgeConnectivity[g] gives the minimum number of edges whose deletion from
   graph g disconnects it. EdgeConnectivity[g, Cut] gives a set of
   edges of minimum size whose deletion disconnects the graph. More...
```

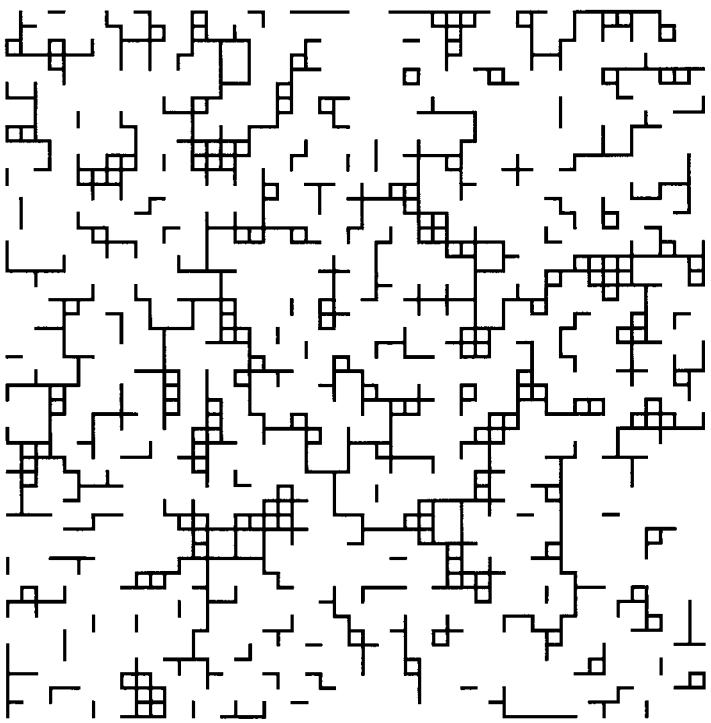
```
The edge-connectivity of graph in fig. 6 is: 1
```

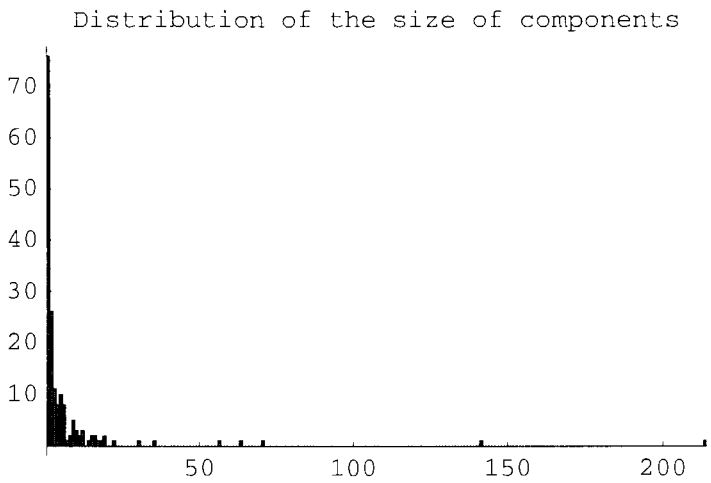
```
? Bridges
Print["The bridges of graph in fig. 6 are: ", Bridges[graphFig6]];
```

```
Bridges[g] gives a list of the bridges of graph g, where each
   bridge is an edge whose removal disconnects the graph. More...
```

```
The bridges of graph in fig. 6 are: {{1, 2}}
```

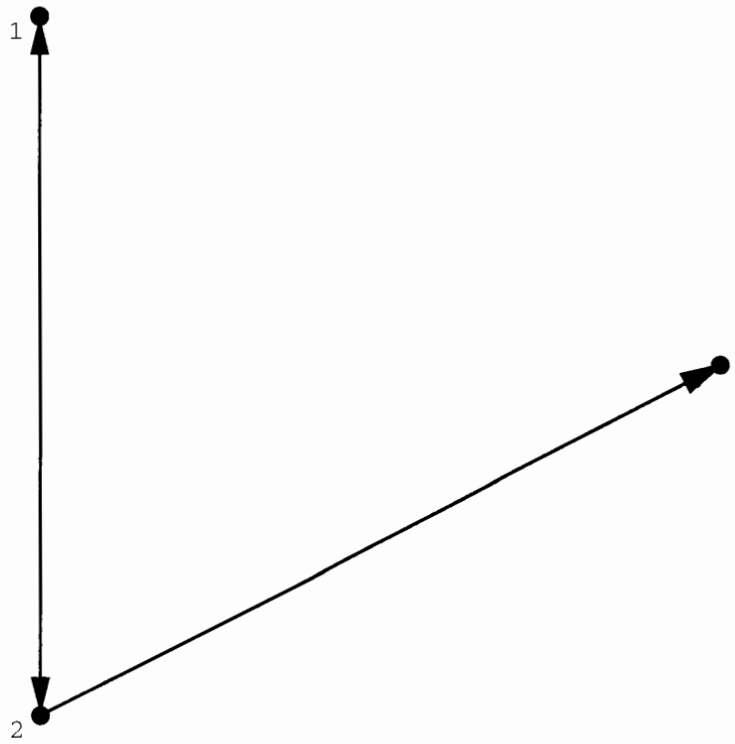
```
ShowGraph[
  myFragmentedGrid = InduceSubgraph[GridGraph[50, 50], RandomSubset[2500]],
  VertexStyle → Disk[0]];
c = Map[Length, ConnectedComponents[myFragmentedGrid]];
Histogram[c, HistogramCategories → Range[0, Max[c] + 1],
  PlotLabel → "Distribution of the size of components"];
```





Local Structures in Networks

```
graphFig8 = EmptyGraph[3, Type → Directed];
graphFig8 = AddEdges[graphFig8, {{1, 2}, {2, 3}, {2, 1}}];
ShowGraph[graphFig8, VertexNumber → True];

Print["Reciprocated relations relative to all possible relations: ",
  ReciprocatedEdges[graphFig8] / (V[graphFig8] * (V[graphFig8] - 1))];
Print["Reciprocated relations relative to existing relations: ",
  ReciprocatedEdges[graphFig8] / M[graphFig8]];
```



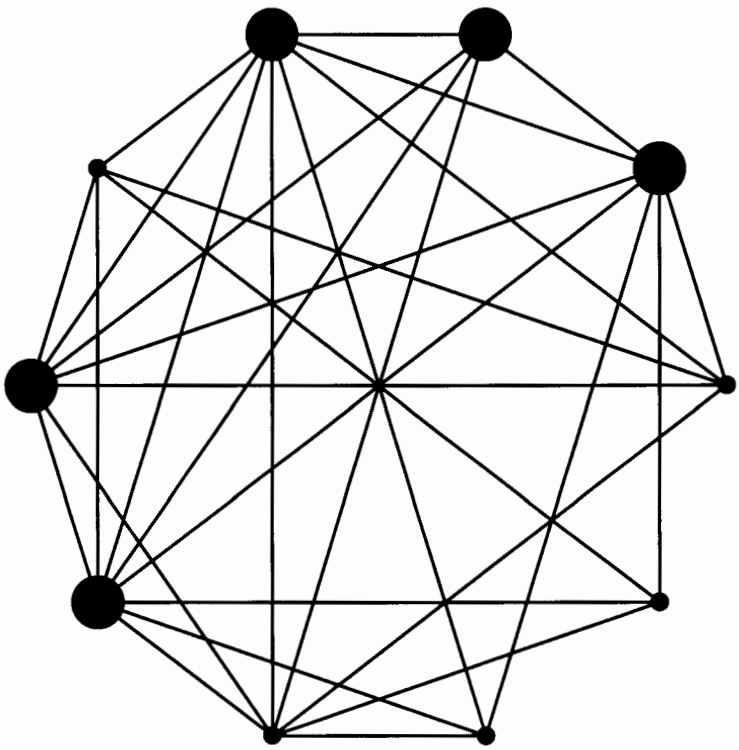
Reciprocated relations relative to all possible relations: $\frac{1}{3}$

Reciprocated relations relative to existing relations: $\frac{2}{3}$

```
? TransitiveQ
? TransitiveClosure
? TransitiveReduction
```

```
TransitiveQ[g] yields True if graph g defines a transitive relation. More...

TransitiveClosure[g] finds the transitive
   closure of graph g, the supergraph of g that contains edge
   {x, y} if and only if there is a path from x to y. More...

TransitiveReduction[g] finds a smallest
   graph that has the same transitive closure as g. More...
```

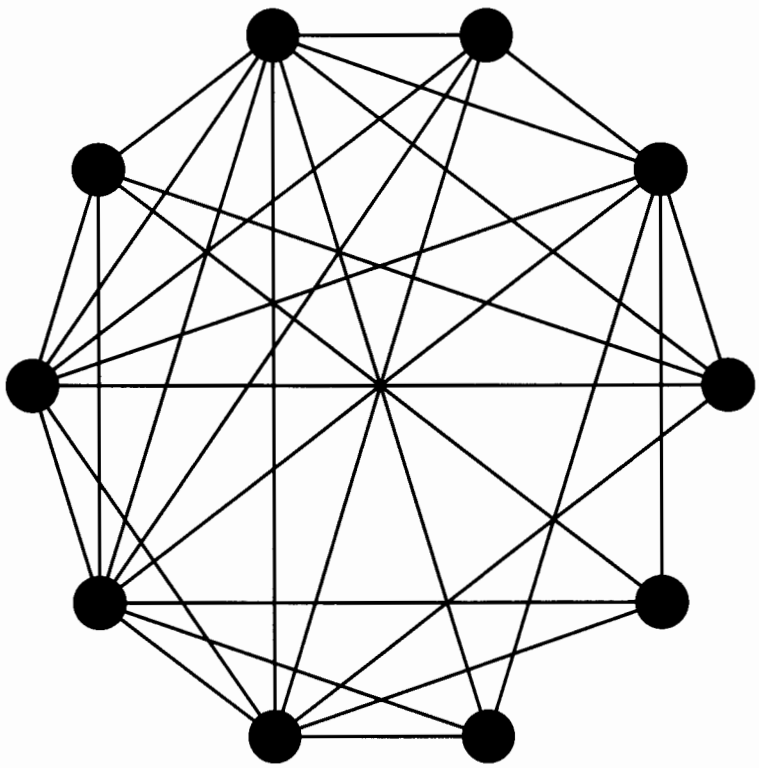
```
SeedRandom[0]
myRandomGraph = RandomGraph[10, 0.7];
ShowGraph[Highlight[myRandomGraph, {MaximumClique[myRandomGraph]}]];
```



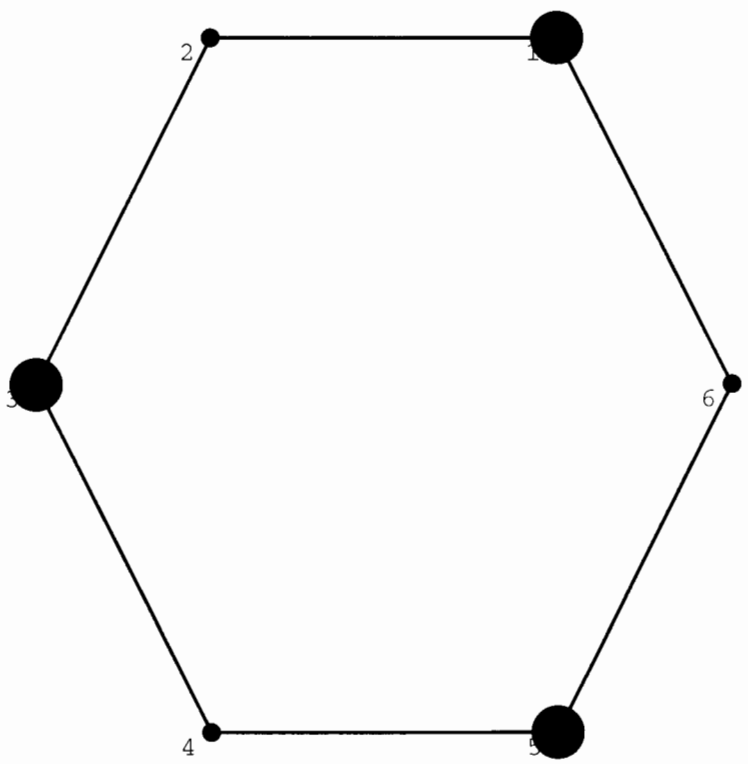
```
Print["Any subset of the vertices of a complete graph forms a clique: ",
  CliqueQ[CompleteGraph[10], RandomSubset[10]]];
```

```
Any subset of the vertices of a complete graph forms a clique: True
```

```
ShowGraph[Highlight[myRandomGraph, {MaximumNClique[myRandomGraph, 2]}]];
```



```
ShowGraph[Highlight[Cycle[6], {1, 3, 5}], VertexNumber → True];
Print["Do nodes 1, 3, and 5 form a 2-Clique? ",
  NCliqueQ[Cycle[6], 2, {1, 3, 5}]];
Print["Do nodes 1, 2, 3, and 5 form a 2-Clique?",
  NCliqueQ[Cycle[6], 2, {1, 2, 3, 5}]];
```




```
Do nodes 1, 3, and 5 form a 2-Clique? True

Do nodes 1, 2, 3, and 5 form a 2-Clique?False
```

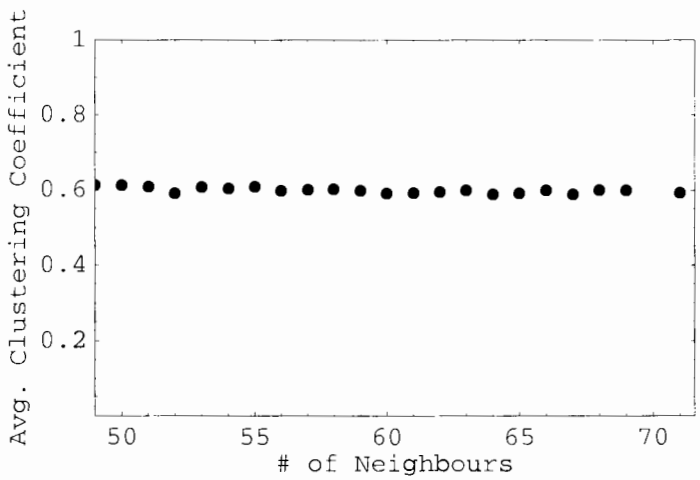
```
Print["Do nodes 1, 3, and 5 form a 2-Clan? ",
  NClanQ[Cycle[6], 2, {1, 3, 5}]];
Print["2-Clans of maximum size in a cycle with 6 nodes: ",
  MaximumNClans[Cycle[6], 2]];
```

```
Do nodes 1, 3, and 5 form a 2-Clan? False

2-Clans of maximum size in a cycle with 6 nodes:
 {{1, 2, 3}, {1, 2, 6}, {1, 5, 6}, {2, 3, 4}, {3, 4, 5}, {4, 5, 6}}
```

```
Print["The clustering coeff. of
    the blue node in the second graph of fig. 9 is: ",
  Clustering[1, DeleteEdge[CompleteGraph[4], {2, 4}]]];
```

The clustering coeff. of the blue node in the second graph of fig. 9 is: $\frac{2}{3}$

```
myListOfClusteringCoefficients = Clustering[RandomGraph[100, 0.6]];
minDegree = Min[myListOfClusteringCoefficients[[All, 1]]];
ListPlot[myListOfClusteringCoefficients,
  AxesOrigin → {minDegree, 0}, PlotRange → {{minDegree, Automatic}, {0, 1}},
  FrameLabel → {"# of Neighbours", "Avg. Clustering Coefficient"},
  Frame → True, PlotStyle → PointSize[0.02]];
```

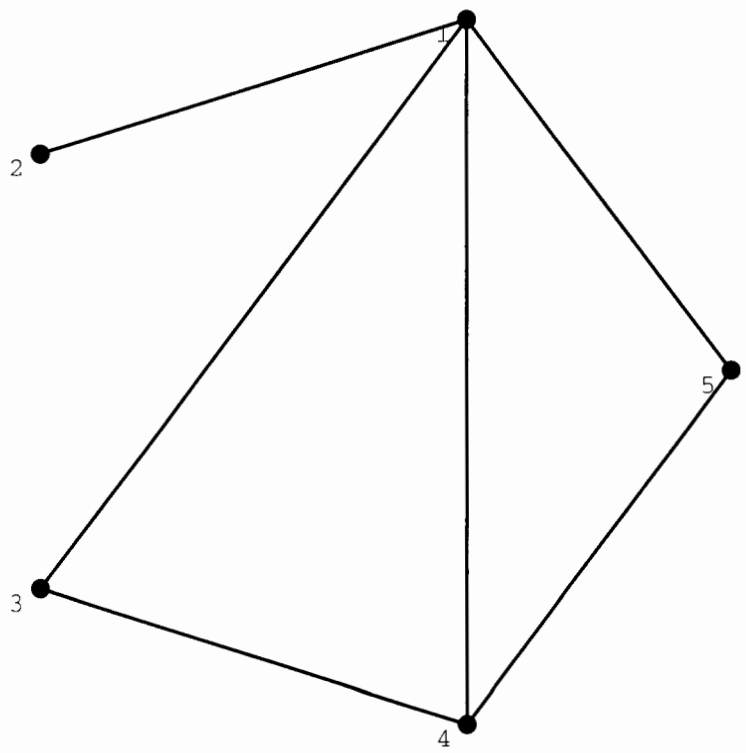



```
Print["The clustering coefficient list of my fragmented grid is: ",
  Clustering[myFragmentedGrid]];
```

The clustering coefficient list of my fragmented grid is:
{{2, 0}, {3, 0}, {4, 0}}

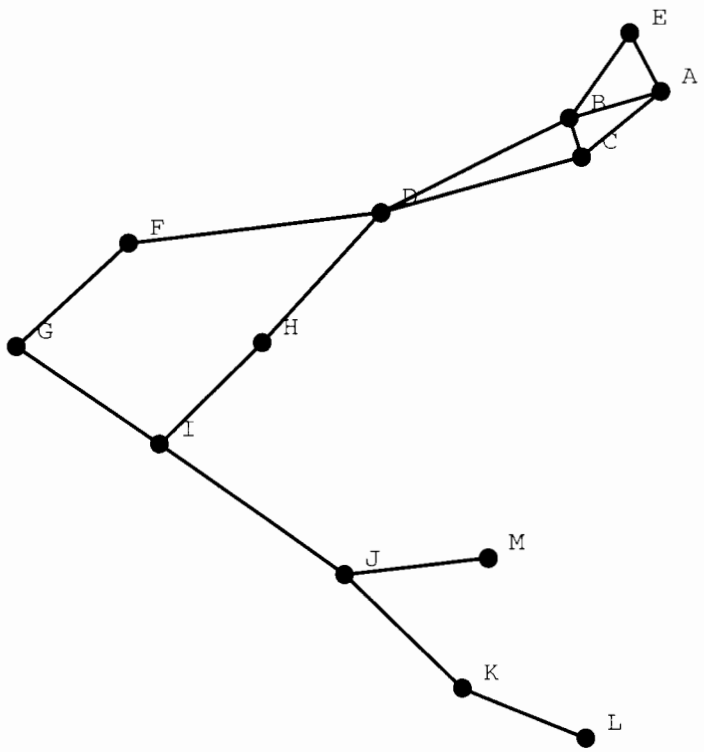
Centrality and Power

```
graphFig10 =
  AddEdges[EmptyGraph[5], {{1, 2}, {1, 3}, {1, 4}, {1, 5}, {3, 4}, {4, 5}}];
ShowGraph[graphFig10, VertexNumber → True];
Print["Geodesic Closeness: ", N[GeodesicCloseness[graphFig10]]];
Print["Reachability Closeness: ", N[ReachabilityCloseness[graphFig10]]];
```



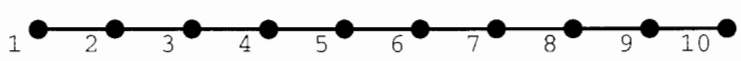

```
Geodesic Closeness: {0.25, 0.142857, 0.166667, 0.2, 0.166667}

Reachability Closeness: {1., 0.625, 0.75, 0.875, 0.75}
```

```
(*
 (*The following code imports an excel file with the adjacency matrix*)
 SetDirectory["U:\MainDirectory\Research\
    InstitutionsAndNetworks\IntroductionToFormalAnalysis"];
 myNetworkMatrix=Import["betweennessNetwork.xls","XLS"];
 nNodes=Length[myNetworkMatrix]-1;
 myAdjMatrix=Round[Take[myNetworkMatrix,-nNodes,-nNodes]];
 vertexLabels=Rest[myNetworkMatrix[[1]]];
 *)

myAdjMatrix = {
   {0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0}, {1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0},
   {1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0},
   {1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0},
   {0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0}, {0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0},
   {0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1},
   {0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 0, 0, 0}};
vertexLabels = {"A", "B", "C", "D", "E", "F", "G",
   "H", "I", "J", "K", "L", "M"};

graphFig12 =
  SetVertexLabels[FromAdjacencyMatrix[myAdjMatrix], vertexLabels];
ShowGraph[SpringEmbedding[graphFig12]];
N[Inner[List, vertexLabels, Betweenness[graphFig12], List]] // MatrixForm
```



A	0.0149254
B	0.25
C	0.12
D	0.546667
E	0.
F	0.08
G	0.0666667
H	0.4
I	0.493333
J	0.426667
K	0.16
L	0.
M	0.

```
g = Path[10];
ShowGraph[g, VertexNumber → True];
Transpose[Inner[List, Range[10], N[Betweenness[g]], List]] // MatrixForm
```



1	2	3	4	5	6	7	8	9	10
0.	0.222222	0.388889	0.5	0.555556	0.555556	0.5	0.388889	0.222222	0.