Network Analysis 8

8.1 Introduction

Network data are generated when we consider relationships between two or more entities in the data, like the highways connecting cities, friendships between people or their phone calls. In recent years, a huge number of network data are being generated and analyzed in different fields. For instance, in sociology there is interest in analyzing blog networks, which can be built based on their citations, to look for divisions in their structures between political orientations. Another example is infectious disease transmission networks, which are built in epidemiological studies to find the best way to prevent infection of people in a territory, by isolating certain areas. Other examples studied in the field of technology include interconnected computer networks or power grids, which are analyzed to optimize their functioning. We also find examples in academia, where we can build co-authorship networks and citation networks to analyze collaborations among Universities.

Structuring data as networks can facilitate the study of the data for different goals; for example, to discover the weaknesses of a structure. That could be the objective of a biologist studying a community of plants and trying to establish which of its properties promote quick transmission of a disease. A contrasting objective would be to find and exploit structures that work efficiently for the transmission of messages across the network. This may be the goal of an advertising agent trying to find the best strategy for spreading publicity.

How to analyze networks and extract the features we want to study are some of the issues we consider in this chapter. In particular, we introduce some basic concepts related with networks, such as connected components, centrality measures, ego-networks, and PageRank. We present some useful Python tools for the analysis of networks and discuss some of the visualization options. In order to motivate and illustrate the concepts, we perform social network analysis using real data. We present a practical case based on a public dataset which consists of a set of interconnected

L. Igual and S. Seguí, *Introduction to Data Science*,
Undergraduate Topics in Computer Science, DOI 10.1007/978-3-319-50017-1_8

Facebook friendship networks. We formulate multiple questions at different levels: the local/member level, the community level, and the global level.

In general, some of the questions we try to solve are the following:

- What type of network are we dealing with?
- Which is the most representative member of the network in terms of being the most connected to the rest of the members?
- Which is the most representative member of the network in terms of being the most circulated on the paths between the rest of the members?
- Which is the most representative member of the network in terms of proximity to the rest of the members?
- Which is the most representative member of the network in terms of being the most accessible from any location in the network?
- There are many ways of calculating the representativeness or importance of a member, each one with a different meaning, so: how can we illustrate them and compare them?
- Are there different communities in the network? If so, how many?
- Does any member of the network belong to more than one community? That is, is there any overlap between the communities? How much overlap? How can we illustrate this overlap?
- Which is the largest community in the network?
- Which is the most dense community (in terms of connections)?
- How can we automatically detect the communities in the network?
- Is there any difference between automatically detected communities and real ones (manually labeled by users)?

8.2 Basic Definitions in Graphs

Graph is the mathematical term used to refer to a network. Thus, the field that studies networks is called *graph theory* and it provides the tools necessary to analyze networks. Leonhard Euler defined the first graph in 1735, as an abstraction of one of the problems posed by mathematicians of the time regarding Konigsberg, a city with two islands created by the River Pregel, which was crossed by seven bridges. The problem was: is it possible to walk through the town of Konigsberg crossing each bridge once and only once? Euler represented the land areas as nodes and the bridges connecting them as edges of a graph and proved that the walk was not possible for this particular graph.

A graph is defined as a set of *nodes*, which are an abstraction of any entities (parts of a city, persons, etc.), and the connecting links between pairs of nodes called *edges* or relationships. The edge between two nodes can be *directed* or *undirected*. A directed edge means that the edge points from one node to the other and not the other way round. An example of a directed relationship is “a person knows another person”. An edge has a direction when person A knows person B, and not the reverse direction

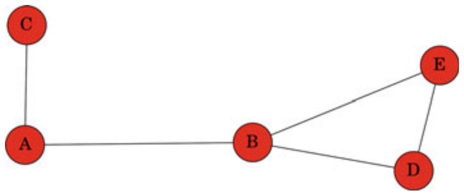

Fig. 8.1 Simple undirected labeled graph with 5 nodes and 5 edges

if B does not know A (which is usual for many fans and celebrities). An undirected edge means that there is a symmetric relationship. An example is "a person shook hands with another person"; in this case, the relationship, unavoidably, involves both persons and there is no directionality. Depending on whether the edges of a graph are directed or undirected, the graph is called a *directed graph* or an *undirected graph*, respectively.

The *degree* of a node is the number of edges that connect to it. Figure 8.1 shows an example of an undirected graph with 5 nodes and 5 edges. The degree of node C is 1, while the degree of nodes A, D and E is 2 and for node B it is 3. If a network is directed, then nodes have two different degrees, the *in-degree*, which is the number of incoming edges, and the *out-degree*, which is the number of outgoing edges.

In some cases, there is information we would like to add to graphs to model properties of the entities that the nodes represent or their relationships. We could add *strengths* or *weights* to the links between the nodes, to represent some real-world measure. For instance, the length of the highways connecting the cities in a network. In this case, the graph is called a *weighted graph*.

Some other elementary concepts that are useful in graph analysis are those we explain in what follows. We define a *path* in a network to be a sequence of nodes connected by edges. Moreover, many applications of graphs require *shortest paths* to be computed. The shortest path problem is the problem of finding a path between two nodes in a graph such that the length of the path or the sum of the weights of edges in the path is minimized. In the example in Fig. 8.1, the paths (C, A, B, E) and (C, A, B, D, E) are those between nodes C and E. This graph is unweighted, so the shortest path between C and E is the one that follows the fewer edges: (C, A, B, E).

A graph is said to be *connected* if for every pair of nodes, there is a path between them. A graph is *fully connected* or *complete* if each pair of nodes is connected by an edge. A *connected component* or simply a *component* of a graph is a subset of its nodes such that every node in the subset has a path to every other one. In the example of Fig. 8.1, the graph has one connected component. A *subgraph* is a subset of the nodes of a graph and all the edges linking those nodes. Any group of nodes can form a subgraph.

8.3 Social Network Analysis

Social network analysis processes social data structured in graphs. It involves the extraction of several characteristics and graphics to describe the main properties of the network. Some general properties of networks, such as the shape of the network degree distribution (defined bellow) or the average path length, determine the type of network, such as a *small-world* network or a *scale-free* network. A small-world network is a type of graph in which most nodes are not neighbors of one another, but most nodes can be reached from every other node in a small number of steps. This is the so-called *small-world phenomenon* which can be interpreted by the fact that strangers are linked by a short chain of acquaintances. In a small-world network, people usually form communities or small groups where everyone knows everyone else. Such communities can be seen as complete graphs. In addition, most the community members have a few relationships with people outside that community. However, some people are connected to a large number of communities. These may be celebrities and such people are considered as the *hubs* that are responsible for the small-world phenomenon. Many small-world networks are also scale-free networks. In a scale-free network the node degree distribution follows a power law (a relationship function between two quantities x and y defined as $y = x^n$, where n is a constant). The name *scale-free* comes from the fact that power laws have the same functional form at all scales, i.e., their shape does not change on multiplication by a scale factor. Thus, by definition, a scale-free network has many nodes with a very few connections and a small number of nodes with many connections. This structure is typical of the World Wide Web and other social networks. In the following sections, we illustrate this and other graph properties that are useful in social network analysis.

8.3.1 Basics in NetworkX

NetworkX[1] is a Python toolbox for the creation, manipulation and study of the structure, dynamics and functions of complex networks. After importing the toolbox, we can create an undirected graph with 5 nodes by adding the edges, as is done in the following code. The output is the graph in Fig. 8.1.

In [1]:
```
import networkx as nx
G = nx.Graph()
G.add_edge('A', 'B');
G.add_edge('A', 'C');
G.add_edge('B', 'D');
G.add_edge('B', 'E');
G.add_edge('D', 'E');
nx.draw_networkx(G)
```

To create a directed graph we would use `nx.DiGraph()`.

[1] https://networkit.iti.kit.edu.

8.3.2 Practical Case: Facebook Dataset

For our practical case we consider data from the Facebook network. In particular, we use the data *Social circles: Facebook*[2] from the Stanford Large Network Dataset[3] (SNAP) collection. The SNAP collection has links to a great variety of networks such as Facebook-style social networks, citation networks, Twitter networks or open communities like Live Journal. The Facebook dataset consists of a network representing friendship between Facebook users. The Facebook data was anonymized by replacing the internal Facebook identifiers for each user with a new value.

The network corresponds to an undirected and unweighted graph that contains users of Facebook (nodes) and their friendship relations (edges). The Facebook dataset is defined by an edge list in a plain text file with one edge per line.

Let us load the Facebook network and start extracting the basic information from the graph, including the numbers of nodes and edges, and the average degree:

In [2]:
```
fb = nx.read_edgelist("files/ch08/facebook_combined.txt")
fb_n, fb_k = fb.order(), fb.size()
fb_avg_deg = fb_k / fb_n
print 'Nodes: ', fb_n
print 'Edges: ', fb_k
print 'Average degree: ', fb_avg_deg
```

Out[2]:
```
Nodes: 4039
Edges: 88234
Average degree: 21
```

The Facebook dataset has a total of 4,039 users and 88,234 friendship connections, with an average degree of 21. In order to better understand the graph, let us compute the degree distribution of the graph. If the graph were directed, we would need to generate two distributions: one for the in-degree and another for the out-degree. A way to illustrate the degree distribution is by computing the histogram of degrees and plotting it, as the following code does with the output shown in Fig. 8.2:

In [3]:
```
degrees = fb.degree().values()
degree_hist = plt.hist(degrees, 100)
```

The graph in Fig. 8.2 is a power-law distribution. Thus, we can say that the Facebook network is a *scale-free network*.

Next, let us find out if the Facebook dataset contains more than one connected component (previously defined in Sect. 8.2):

In [4]:
```
print '# connected components of Facebook network: ',
      nx.number_connected_components(fb)
```

Out[4]:
```
# connected components of Facebook network: 1
```

[2]https://snap.stanford.edu/data/egonets-Facebook.html.
[3]http://snap.stanford.edu/data/.

As it can be seen, there is only one connected component in the Facebook network. Thus, the Facebook network is a connected graph (see definition in Sect. 8.2). We can try to divide the graph into different connected components, which can be potential communities (see Sect. 8.6). To do that, we can remove one node from the graph (this operation also involves removing the edges linking the node) and see if the number of connected components of the graph changes. In the following code, we prune the graph by removing node ‘0’ (arbitrarily selected) and compute the number of connected components of the pruned version of the graph:

In [5]:
```
fb_prun = nx.read_edgelist(
    "files/ch08/facebook_combined.txt")
fb_prun.remove_node('0')
print 'Remaining nodes:', fb_prun.number_of_nodes()
print 'New # connected components:',
      nx.number_connected_components(fb_prun)
```

Out[5]:
```
Remaining nodes: 4038
New # connected components: 19
```

Now there are 19 connected components, but let us see how big the biggest is and how small the smallest is:

In [6]:
```
fb_components = nx.connected_components(fb_prun)
print 'Sizes of the connected components',
      [len(c) for c in fb_components]
```

Out[6]:
```
Sizes of the connected components [4015, 1, 3, 2, 2, 1, 1, 1,
1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1]
```

This simple example shows that removing a node splits the graph into multiple components. You can see that there is one large connected component and the rest are almost all isolated nodes. The isolated nodes in the pruned graph were only

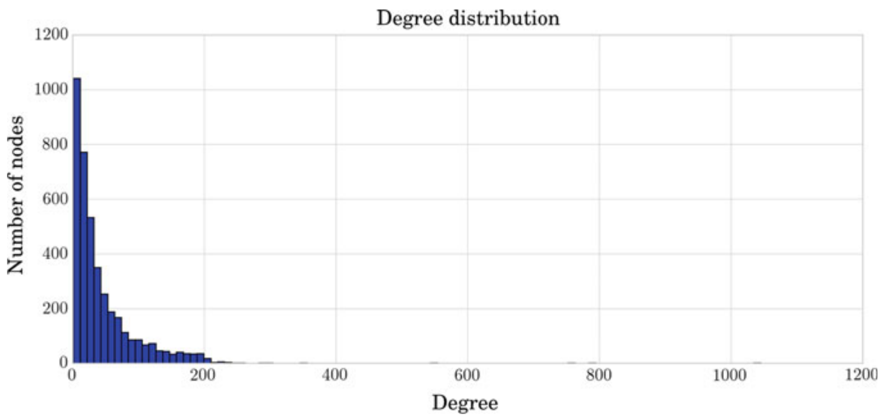

Fig. 8.2 Degree histogram distribution

connected to node '0' in the original graph and when that node was removed they were converted into connected components of size 1. These nodes, only connected to one neighbor, are probably not important nodes in the structure of the graph. We can generalize the analysis by studying the *centrality* of the nodes. The next section is devoted to explore this concept.

8.4 Centrality

The centrality of a node measures its relative importance within the graph. In this section we focus on undirected graphs. Centrality concepts were first developed in social network analysis. The first studies indicated that central nodes are probably more influential, have greater access to information, and can communicate their opinions to others more efficiently [1]. Thus, the applications of centrality concepts in a social network include identifying the most influential people, the most informed people, or the most communicative people. In practice, what centrality means will depend on the application and the meaning of the entities represented as nodes in the data and the connections between those nodes. Various measures of the centrality of a node have been proposed. We present four of the best-known measures: *degree centrality*, *betweenness centrality*, *closeness centrality*, and *eigenvector centrality*.

Degree centrality is defined as the number of edges of the node. So the more ties a node has, the more central the node is. To achieve a normalized degree centrality of a node, the measure is divided by the total number of graph nodes (n) without counting this particular one ($n - 1$). The normalized measure provides proportions and allows us to compare it among graphs. Degree centrality is related to the capacity of a node to capture any information that is floating through the network. In social networks, connections are associated with positive aspects such as knowledge or friendship.

Betweenness centrality quantifies the number of times a node is crossed along the shortest path/s between any other pair of nodes. For the normalized measure this number is divided by the total number of shortest paths for every pair of nodes. Intuitively, if we think of a public bus transportation network, the bus stop (node) with the highest betweenness has the most traffic. In social networks, a person with high betweenness has more power in the sense that more people depend on him/her to make connections with other people or to access information from other people. Comparing this measure with degree centrality, we can say that degree centrality depends only on the node's neighbors; thus, it is more local than the betweenness centrality, which depends on the connection properties of every pair of nodes in the graph, except pairs with the node in question itself. The equivalent measure exists for edges. The betweenness centrality of an edge is the proportion of the shortest paths between all node pairs which pass through it.

Closeness centrality tries to quantify the position a node occupies in the network based on a distance calculation. The distance metric used between a pair of nodes is defined by the length of its shortest path. The closeness of a node is inversely proportional to the length of the average shortest path between that node and all the other nodes in the graph. In this case, we interpret a central node as being close to, and able to communicate quickly with, the other nodes in a social network.

Eigenvector centrality defines a relative score for a node based on its connections and considering that connections from high centrality nodes contribute more to the score of the node than connections from low centrality nodes. It is a measure of the influence of a node in a network, in the following sense: it measures the extent to which a node is connected to influential nodes. Accordingly, an important node is connected to important neighbors.

Let us illustrate the centrality measures with an example. In Fig. 8.3, we show an undirected star graph with $n = 8$ nodes. Node C is obviously important, since it can exchange information with more nodes than the others. The degree centrality measures this idea. In this star network, node C has a degree centrality of 7 or 1 if we consider the normalized measure, whereas all other nodes have a degree of 1 or 1/7 if we consider the normalized measure. Another reason why node C is more important than the others in this star network is that it lies between each of the other pairs of nodes, and no other node lies between C and any other node. If node C wants to contact F, C can do it directly; whereas if node F wants to contact B, it must go through C. This gives node C the capacity to broke/prevent contact among other nodes and to isolate nodes from information. The betweenness centrality is underneath this idea. In this example, the betweenness centrality of the node C is 28, computed as $(n-1)(n-2)/2$, while the rest of nodes have a betweenness of 0. The final reason why we can say node C is superior in the star network is because C is closer to more nodes than any other node is. In the example, node C is at a distance of 1 from all other 7 nodes and each other node is at a distance 2 from all other nodes, except C. So, node C has closeness centrality of 1/7, while the rest of nodes have a closeness of 1/13. The normalized measures, computed by dividing by $n - 1$, are 1 for C and 7/13 for the other nodes.

An important concept in social network analysis is that of a *hub* node, which is defined as a node with high degree centrality and betweenness centrality. When a hub governs a very centralized network, the network can be easily fragmented by removing that hub.

Coming back to the Facebook example, let us compute the degree centrality of Facebook graph nodes. In the code below we show the user identifier of the 10 most central nodes together with their normalized degree centrality measure. We also show the degree histogram to extract some more information from the shape of the distribution. It might be useful to represent distributions using logarithmic scale. We

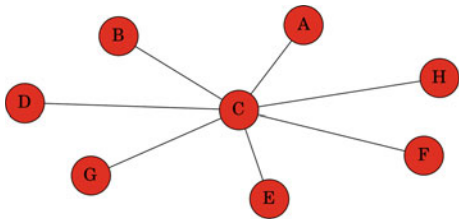

Fig. 8.3 Star graph example

do that with the `matplotlib.loglog()` function. Figure 8.4 shows the degree centrality histogram in linear and logarithmic scales as computed in the box bellow.

```
In [7]:
degree_cent_fb = nx.degree_centrality(fb)
print 'Facebook degree centrality: ',
      sorted(degree_cent_fb.items(),
             key = lambda x: x[1],
             reverse = True)[:10]
degree_hist = plt.hist(list(degree_cent_fb.values()), 100)
plt.loglog(degree_hist[1][1:],
           degree_hist[0], 'b', marker = 'o')
```

```
Out[7]: Facebook degree centrality: [(u'107', 0.258791480931154),
(u'1684', 0.1961367013372957), (u'1912', 0.18697374938088163),
(u'3437', 0.13546310054482416), (u'0', 0.08593363051015354),
(u'2543', 0.07280832095096582), (u'2347', 0.07206537890044576),
(u'1888', 0.0629024269440317), (u'1800', 0.06067360079247152),
(u'1663', 0.05819712729073798 4)]
```

The previous plots show us that there is an interesting (large) set of nodes which corresponds to low degrees. The representation using a logarithmic scale (right-hand graphic in Fig. 8.4) is useful to distinguish the members of this set of nodes, which are clearly visible as a straight line at low values for the x-axis (upper left-hand part of the logarithmic plot). We can conclude that most of the nodes in the graph have low degree centrality; only a few of them have high degree centrality. These latter nodes can be properly seen as the points in the bottom right-hand part of the logarithmic plot.

The next code computes the betweenness, closeness, and eigenvector centrality and prints the top 10 central nodes for each measure.

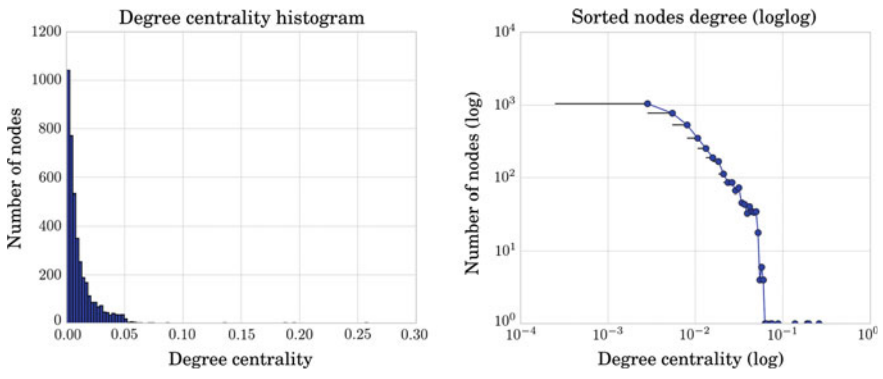

Fig. 8.4 Degree centrality histogram shown using a linear scale (*left*) and a log scale for both the x- and y-axis (*right*)

In [8]:
```
betweenness_fb = nx.betweenness_centrality(fb)
closeness_fb = nx.closeness_centrality(fb)
eigencentrality_fb = nx.eigenvector_centrality(fb)
print 'Facebook betweenness centrality:',
      sorted(betweenness_fb.items(),
             key = lambda x: x[1],
             reverse = True)[:10]
print 'Facebook closeness centrality:',
      sorted(closeness_fb.items(),
             key = lambda x: x[1],
             reverse = True)[:10]
print 'Facebook eigenvector centrality:',
      sorted(eigencentrality_fb.items(),
             key = lambda x: x[1],
             reverse = True)[:10]
```

Out[8]:
```
Facebook betweenness centrality: [(u'107', 0.4805180785560141),
(u'1684', 0.3377974497301984 3), (u'3437', 0.23611535735892616),
(u'1912', 0.2292953395868727), (u'1085', 0.149015092116652 6),
(u'0', 0.1463059214744276), (u'698', 0.11533045020560861),
(u'567', 0.09631033121856114), (u'58', 0.08436020590796521),
(u'428', 0.06430906239323908)]
```

Out[8]:
```
Facebook closeness centrality: [(u'107', 0.45969945355191255),
(u'58', 0.3974018305284913), (u'428', 0.3948371956585509),
(u'563', 0.3939127889961955), (u'1684', 0.39360561458231796),
(u'171', 0.37049270575282134), (u'348', 0.36991572004397216),
(u'483', 0.3698479575013739), (u'414', 0.3695433330282786),
(u'376', 0.36655773420479304)]
Facebook eigenvector centrality: [(u'1912', 0.09540688873596524),
(u'2266', 0.08698328226321951), (u'2206', 0.08605240174265624),
(u'2233', 0.08517341350597836), (u'2464', 0.08427878364685948),
(u'2142', 0.08419312450068105), (u'2218', 0.08415574433673866),
(u'2078', 0.08413617905810111), (u'2123', 0.08367142125897363),
(u'1993', 0.08353243711860482)]
```

As can be seen in the previous results, each measure gives a different ordering of the nodes. The node '107' is the most central node for degree (see box `Out [7]`), betweenness, and closeness centrality, while it is not among the 10 most central nodes for eigenvector centrality. The second most central node is different for closeness and eigenvector centralities; while the third most central node is different for all four centrality measures.

Another interesting measure is the *current flow betweenness centrality*, also called *random walk betweenness centrality*, of a node. It can be defined as the probability of passing through the node in question on a random walk starting and ending at some node. In this way, the betweenness is not computed as a function of shortest paths, but of all paths. This makes sense for some social networks where messages may get to their final destination not by the shortest path, but by a random path, as in the case of gossip floating through a social network for example.

Computing the current flow betweenness centrality can take a while, so we will work with a trimmed Facebook network instead of the original one. In fact, we can pose the question: What happen if we only consider the graph nodes with more than the average degree of the network (21)? We can trim the graph using degree centrality values. To do this, in the next code, we define a function to trim the graph based on the degree centrality of the graph nodes. We set the threshold to 21 connections:

In [9]:
```
def trim_degree_centrality(graph, degree = 0.01):
    g = graph.copy()
    d = nx.degree_centrality(g)
    for n in g.nodes():
        if d[n] <= degree:
            g.remove_node(n)
    return g
thr = 21.0/(fb.order() - 1.0)

print 'Degree centrality threshold:', thr

fb_trimmed = trim_degree_centrality(fb, degree = thr)
print 'Remaining # nodes:', len(fb_trimmed)
```

Out[9]:
```
Degree centrality threshold: 0.00520059435364
Remaining # nodes: 2226
```

The new graph is much smaller; we have removed almost half of the nodes (we have moved from 4,039 to 2,226 nodes).

The current flow betweenness centrality measure needs connected graphs, as does any betweenness centrality measure, so we should first extract a connected component from the trimmed Facebook network and then compute the measure:

In [10]:
```
fb_subgraph = list(nx.connected_component_subgraphs(
    fb_trimed))
print '# subgraphs found:', size(fb_subgraph)
print '# nodes in the first subgraph:',
      len(fb_subgraph[0])
betweenness = nx.betweenness_centrality(fb_subgraph[0])
print 'Trimmed FB betweenness: ',
      sorted(betweenness.items(), key = lambda x: x[1],
             reverse = True)[:10]
current_flow = nx.current_flow_betweenness_centrality(
    fb_subgraph[0])
print 'Trimmed FB current flow betweenness:',
      sorted(current_flow.items(), key = lambda x: x[1],
      reverse = True)[:10]
```

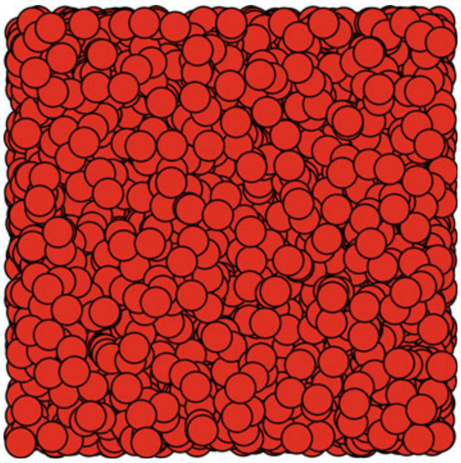
Fig. 8.5 The Facebook network with a random layout

```
Out[10]: # subgraphs found: 2
         # nodes in the first subgraph: 2225
         Trimmed FB betweenness: [(u'107', 0.5469164906683255),
         (u'1684', 0.3133966633778371), (u'1912', 0.19965597457246995),
         (u'3437', 0.13002843874261014), (u'1577', 0.1274607407928195),
         (u'1085', 0.11517250980098293), (u'1718', 0.08916631761105698),
         (u'428', 0.0638271827912378), (u'1465', 0.057995900747731755),
         (u'567', 0.05414376521577943)]
         Trimmed FB current flow betweenness: [(u'107',
         0.2858892136334576), (u'1718', 0.2678396761785764), (u'1684',
         0.1585162194931393), (u'1085', 0.1572155780323929), (u'1405',
         0.1253563113363113), (u'3437', 0.10482568101478178), (u'1912',
         0.09369897700970155), (u'1577', 0.08897207040045449), (u'136',
         0.07052866082249776), (u'1505', 0.06152347046861114)]
```

As can be seen, there are similarities in the 10 most central nodes for the betweenness and current flow betweenness centralities. In particular, seven up to ten are the same nodes, even if they are differently ordered.

8.4.1 Drawing Centrality in Graphs

In this section we focus on graph visualization, which can help in the network data understanding and usability.

The visualization of a network with a large amount of nodes is a complex task. Different layouts can be used to try to build a proper visualization. For instance, we can draw the Facebook graph using the random layout (`nx.random_layout`), but this is a bad option, as can be seen in Fig. 8.5. Other alternatives can be more useful. In the box below, we use the *Spring* layout, as it is used in the default function (`nx.draw`), but with more iterations. The function `nx.spring_layout` returns the position of the nodes using the Fruchterman–Reingold force-directed algorithm.

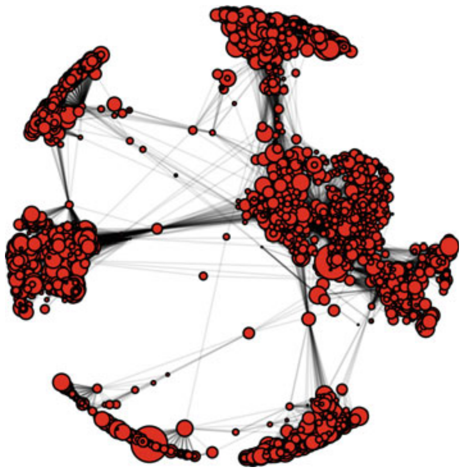

Fig. 8.6 The Facebook network drawn using the Spring layout and degree centrality to define the node size

This algorithm distributes the graph nodes in such a way that all the edges are more or less equally long and they cross themselves as few times as possible. Moreover, we can change the size of the nodes to that defined by their degree centrality. As can be seen in the code, the degree centrality is normalized to values between 0 and 1, and multiplied by a constant to make the sizes appropriate for the format of the figure:

In [11]:
```
pos_fb = nx.spring_layout(fb,iterations = 1000)

nsize = np.array([v for v in degree_cent_fb.values()])

nsize = 500*(nsize - min(nsize))/(max(nsize) - min(nsize))

nodes = nx.draw_networkx_nodes(fb, pos = pos_fb,
                               node_size = nsize)
edges = nx.draw_networkx_edges(fb, pos = pos_fb,
                               alpha = .1)
```

The resulting graph visualization is shown in Fig. 8.6. This illustration allows us to understand the network better. Now we can distinguish several groups of nodes or "communities" clearly in the graph. Moreover, the larger nodes are the more central nodes, which are highly connected of the Facebook graph.

We can also use the betweenness centrality to define the size of the nodes. In this way, we obtain a new illustration stressing the nodes with higher betweenness, which are those with a large influence on the transfer of information through the network. The new graph is shown in Fig. 8.7. As expected, the central nodes are now those connecting the different communities.

Generally different centrality metrics will be positively correlated, but when they are not, there is probably something interesting about the network nodes. For instance, if you can spot nodes with high betweenness but relatively low degree, these are the nodes with few links but which are crucial for network flow. We can also look for

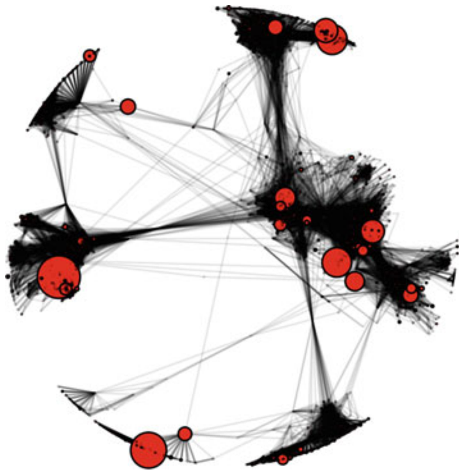

Fig. 8.7 The Facebook network drawn using the Spring layout and betweenness centrality to define the node size

the opposite effect: nodes with high degree but relatively low betweenness. These nodes are those with redundant communication.

Changing the centrality measure to closeness and eigenvector, we obtain the graphs in Figs. 8.8 and 8.9, respectively. As can be seen, the central nodes are also different for these measures. With this or other visualizations you will be able to discern different types of nodes. You can probably see nodes with high closeness centrality but low degree; these are essential nodes linked to a few important or active nodes. If the opposite occurs, if there are nodes with high degree centrality but low closeness, these can be interpreted as nodes embedded in a community that is far removed from the rest of the network.

In other examples of social networks, you could find nodes with high closeness centrality but low betweenness; these are nodes near many people, but since there may be multiple paths in the network, they are not the only ones to be near many people. Finally, it is usually difficult to find nodes with high betweenness but low closeness, since this would mean that the node in question monopolized the links from a small number of people to many others.

8.4.2 PageRank

PageRank is an algorithm related to the concept of eigenvector centrality in directed graphs. It is used to rate webpages objectively and effectively measure the attention devoted to them. PageRank was invented by Larry Page and Sergey Brin, and became a Google trademark in 1998 [2].

Assigning the importance of a webpage is a subjective task, which depends on the interests and knowledge of the persons that browse the webpages. However, there are ways to objectively rank the relative importance of webpages.

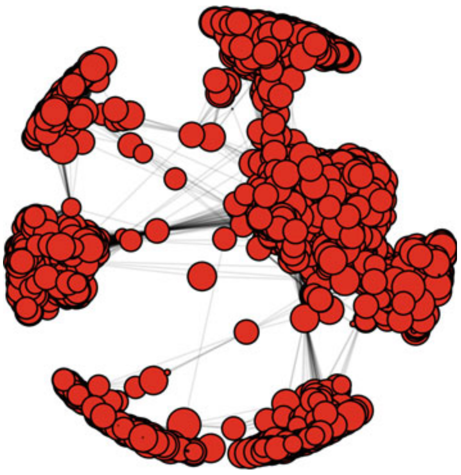
Fig. 8.8 The Facebook network drawn using the Spring layout and closeness centrality to define the node size

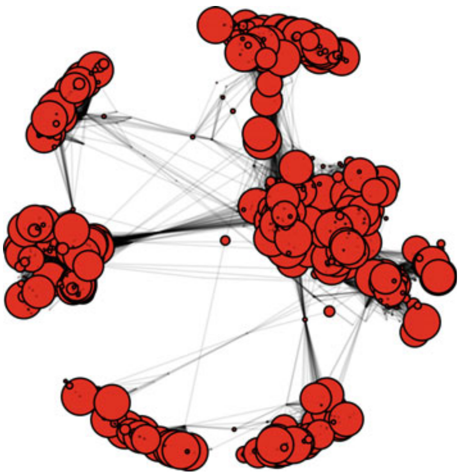
Fig. 8.9 The Facebook network drawn using the Spring layout and eigenvector centrality to define the node size

We consider the directed graph formed by nodes corresponding to the webpages and edges corresponding to the hyperlinks. Intuitively, a hyperlink to a page counts as a vote of support and a page has a high rank if the sum of the ranks of its incoming edges is high. This considers both cases when a page has many incoming links and when a page has a few highly ranked incoming links. Nowadays, a variant of the algorithm is used by Google. It does not only use information on the number of edges pointing into and out of a website, but uses many more variables.

We can describe the PageRank algorithm from a probabilistic point of view. The rank of page P_i is the probability that a surfer on the Internet who starts visiting a random page and follows links, visits the page P_i. With more details, we consider that the weights assigned to the edges of a network by its transition matrix, M, are the probabilities that the surfer goes from one webpage to another. We can understand the

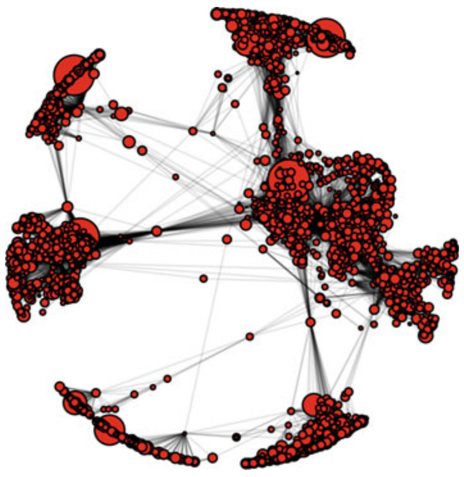

Fig. 8.10 The Facebook network drawn using the Spring layout and PageRank to define the node size

rank computation as a random walk through the network. We start with an initial equal probability for each page: $v_0 = (\frac{1}{n}, \dots, \frac{1}{n})$, where n is the number of nodes. Then we can compute the probability that each page is visited after one step by applying the transition matrix: $v_1 = Mv$. The probability that each page will be visited after k steps is given by $v_k = M^k a$. After several steps, the sequence converges to a unique probabilistic vector a^* which is the *PageRank vector*. The i-th element of this vector is the probability that at each moment the surfer visits page P_i. We need a nonambiguous definition of the rank of a page for any directed web graph. However, in the Internet, we can expect to find pages that do not contain outgoing links and this configuration can lead to certain problems to the explained procedure. In order to overcome this problem, the algorithm fixes a positive constant p between 0 and 1 (a typical value for p is 0.85) and redefines the transition matrix of the graph by $R = (1 - p) M + p B$, where $B = \frac{1}{n} I$, and I is the identity matrix. Therefore, a node with no outgoing edges has probability $\frac{p}{n}$ of moving to any other node.

Let us compute the PageRank vector of the Facebook network and use it to define the size of the nodes, as was done in box `In [11]`.

In [12]:
```
pr = nx.pagerank(fb, alpha = 0.85)
nsize = np.array([v for v in pr.values()])
nsize = 500*(nsize  - min(nsize))/(max(nsize) - min(nsize))
nodes = nx.draw_networkx_nodes(fb,
                               pos = pos_fb,
                               node_size = nsize)
edges = nx.draw_networkx_edges(fb,
                               pos = pos_fb,
                               alpha = .1)
```

The code above outputs the graph in Fig. 8.10, that emphasizes some of the nodes with high PageRank. Looking the graph carefully one can realize that there is one large node per community.

8.5 Ego-Networks

Ego-networks are subnetworks of neighbors that are centered on a certain node. In Facebook and LinkedIn, these are described as “your network". Every person in an ego-network has her/his own ego-network and can only access the nodes in it. All ego-networks interlock to form the whole social network. The ego-network definition depends on the network distance considered. In the basic case, a distance of 1, a link means that person A is a friends of person B, a distance of 2 means that a person, C, is a friend of a friend of A, and a distance of 3 means that another person, D, is a friend of a friend of a friend of A. Knowing the size of an ego-network is important when it comes to understanding the reach of the information that a person can transmit or have access to. Figure 8.11 shows an example of an ego-network. The blue node is the *ego*, while the rest of the nodes are red.

Our Facebook network was manually labeled by users into a set of 10 ego-networks. The public dataset includes the information of these 10 manually defined ego-networks. In particular, we have available the list of the 10 ego nodes: ‘0’, ‘107’, ‘348’, ‘414’, ‘686’, ‘1684’, ‘1912’, ‘3437’, ‘3980’ and their connections. These ego-networks are interconnected to form the fully connected graph we have been analyzing in previous sections.

In Sect. 8.4 we saw that node ‘107’ is the most central node of the Facebook network for three of the four centrality measures computed. So, let us extract the ego-networks of the popular node ‘107’ with a distance of 1 and 2, and compute their sizes. NetworkX has a function devoted to this task:

In [13]:
```
ego_107 = nx.ego_graph(fb, '107')
print '# nodes of ego graph 107:',
      len(ego_107)
print '# nodes of ego graph 107 with radius up to 2:',
      len(nx.ego_graph(fb, '107', radius = 2))
```

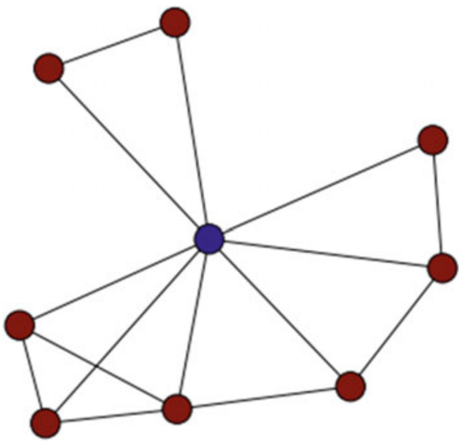
Fig. 8.11 Example of an ego-network. The *blue* node is the ego

Out[13]:
```
# nodes of ego graph 107: 1046
# nodes of ego graph 107 with radius up to 2: 2687
```

The ego-network size is 1,046 with a distance of 1, but when we expand the distance to 2, node '107' is able to reach up to 2,687 nodes. That is quite a large ego-network, containing more than half of the total number of nodes.

Since the dataset also provides the previously labeled ego-networks, we can compute the actual size of the ego-network following the user labeling. We can access the ego-networks by simply importing `os.path` and reading the edge list corresponding, for instance, to node '107', as in the following code:

In [14]:
```
import os.path
ego_id = 107
G_107 = nx.read_edgelist(
    os.path.join('files/ch08/facebook',
                 '{0}.edges'.format(ego_id)),
    nodetype = int)
print 'Nodes of the ego graph 107:', len(G_107)
```

Out[14]:
```
Nodes of the ego graph 107: 1034
```

As can be seen, the size of the previously defined ego-network of node '107' is slightly different from the ego-network automatically computed using NetworkX. This is due to the fact that the manual labeling is not necessarily referred to the subgraph of neighbors at a distance of 1.

We can now answer some other questions about the structure of the Facebook network and compare the 10 different ego-networks among them. First, we can compute which the most densely connected ego-network is from the total of 10. To do that, in the code below, we compute the number of edges in every ego-network and select the network with the maximum number:

In [15]:
```
ego_ids = ( 0, 107, 348,
            414, 686, 698,
            1684, 1912, 3437, 3980)
ego_sizes = zeros((10, 1))
i = 0
# Fill the 'ego_sizes' vector with the size (# edges) of the
     10 ego-networks in egoids
for id in ego_ids :
    G = nx.read_edgelist(
        os.path.join('files/ch08/facebook',
                     '{0}.edges'.format(id)),
        nodetype = int)
    ego_sizes[i] = G.size()
    i = i + 1
[i_max,j] = (ego_sizes == ego_sizes.max()).nonzero()
ego_max = ego_ids[i_max]
print 'The most densely connected ego-network is \
      that of node:', ego_max

G = nx.read_edgelist(
    os.path.join('files/ch08/facebook',
                 '{0}.edges'.format(ego_max)),
    nodetype = int)
print 'Nodes: ', G.order()
print 'Edges: ', G.size()
print 'Average degree: ', G_k / G_n
```

Out[15]:
```
The most densely connected ego-network is that of node: 1912
Nodes: 747
Edges: 30025
Average degree: 40
```

The most densely connected ego-network is that of node '1912', which has an average degree of 40. We can also compute which is the largest (in number of nodes) ego-network, changing the measure of sizes from `G.size()` by `G.order()`. In this case, we obtain that the largest ego-network is that of node '107', which has 1,034 nodes and an average degree of 25.

Next let us work out how much intersection exists between the ego-networks in the Facebook network. To do this, in the code below, we add a field 'egonet' for every node and store an array with the ego-networks the node belongs to. Then, having the length of these arrays, we compute the number of nodes that belong to 1, 2, 3, 4 and more than 4 ego-networks:

In [16]:
```
# Add a field 'egonet' to the nodes of the whole facebook
    network.
# Default value egonet = [], meaning that this node does not
     belong to any ego-netowrk
for i in fb.nodes() :
    fb.node[str(i)]['egonet'] = []

# Fill the 'egonet' field with one of the 10 ego values in
    ego_ids:
for id in ego_ids :
    G = nx.read_edgelist(
        os.path.join('files/ch08/facebook',
                     '{0}.edges'.format(id)),
        nodetype = int)
    print id
    for n in G.nodes() :
        if (fb.node[str(n)]['egonet'] == []) :
            fb.node[str(n)]['egonet'] = [id]
        else :
            fb.node[str(n)]['egonet'].append(id)

# Compute the intersections:
S = [len(x['egonet']) for x in fb.node.values()]
print '# nodes into 0 ego-network: ', sum(equal(S, 0))
print '# nodes into 1 ego-network: ', sum(equal(S, 1))
print '# nodes into 2 ego-network: ', sum(equal(S, 2))
print '# nodes into 3 ego-network: ', sum(equal(S, 3))
print '# nodes into 4 ego-network: ', sum(equal(S, 4))
print '# nodes into more than 4 ego-network: ',\
      sum(greater(S, 4))
```

Out[16]:
```
# nodes into 0 ego-network: 80
# nodes into 1 ego-network: 3844
# nodes into 2 ego-network: 102
# nodes into 3 ego-network: 11
# nodes into 4 ego-network: 2
# nodes into more than 4 ego-network: 0
```

As can be seen, there is an intersection between the ego-networks in the Facebook network, since some of the nodes belong to more than 1 and up to 4 ego-networks simultaneously.

We can also try to visualize the different ego-networks. In the following code, we draw the ego-networks using different colors on the whole Facebook network and we obtain the graph in Fig. 8.12. As can be seen, the ego-networks clearly form groups of nodes that can be seen as communities.

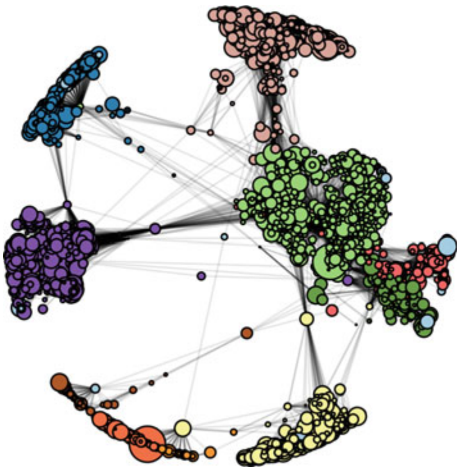
Fig. 8.12 The Facebook network drawn using the Spring layout and different colors to separate the ego-networks

In [17]:
```
# Add a field 'egocolor' to the nodes of the whole facebook
network.
# Default value egocolor r =0, meaning that this node
does not belong to any ego-netowrk for i in fb.nodes() :
    fb.node[str(i)]['egocolor'] = 0

# Fill the 'egocolor' field with a different color number
    for each ego-network in ego_ids:
idColor = 1
for id in ego_ids :
    G = nx.read_edgelist(
        os.path.join('files/ch08/facebook',
                     '{0}.edges'.format(id)),
        nodetype = int)
    for n in G.nodes() :
        fb.node[str(n)]['egocolor'] = idColor
    idColor += 1

colors = [x['egocolor'] for x in fb.node.values()]

nsize = np.array([v for v in degree_cent_fb.values()])

nsize = 500*(nsize - min(nsize))/(max(nsize)- min(nsize))

nodes = nx.draw_networkx_nodes(
    fb, pos = pos_fb,
    cmap = plt.get_cmap('Paired'),
    node_color = colors,
    node_size = nsize,
    with_labels = False)
edges=nx.draw_networkx_edges(fb, pos = pos_fb, alpha = .1)
```

However, the graph in Fig. 8.12 does not illustrate how much overlap is there between the ego-networks. To do that, we can visualize the intersection between ego-networks using a *Venn* or an *Euler* diagram. Both diagrams are useful in order to see how networks are related. Figure 8.13 shows the Venn diagram of the Facebook network. This powerful and complex graph cannot be easily built in Python tool-

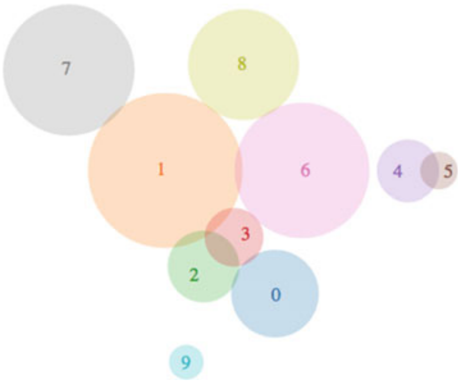

Fig. 8.13 Venn diagram. The area is weighted according to the number of friends in each ego-network and the intersection between ego-networks is related to the number of common users

boxes like NetworkX or Matplotlib. In order to create it, we have used a JavaScript visualization library called D3.JS.[4]

8.6 Community Detection

A community in a network can be seen as a set of nodes of the network that is densely connected internally. The detection of communities in a network is a difficult task since the number and sizes of communities are usually unknown [3].

Several methods for community detection have been developed. Here, we apply one of the methods to automatically extract communities from the Facebook network. We import the `Community` toolbox[5] which implements the Louvain method for community detection. In the code below, we compute the best partition and plot the resulting communities in the whole Facebook network with different colors, as we did in box `In [17]`. The resulting graph is shown in Fig. 8.14.

In [18]:
```
import community partition = community.best_partition(fb)
    print "#
communities found:", max(partition.values()) colors2 =
[partition.get(node) for node in fb.nodes()] nsize = np.
    array([v
for v in degree_cent_fb.values()]) nsize = 500*(nsize  -
min(nsize))/(max(nsize)- min(nsize)) nodes =
nx.draw_networkx_nodes(
    fb, pos = pos_fb,
    cmap = plt.get_cmap('Paired'),
    node_color = colors2,
    node_size = nsize,
    with_labels = False)
edges = nx.draw_networkx_edges(fb, pos = pos_fb, alpha = .1)
```

[4]https://d3js.org.
[5]http://perso.crans.org/aynaud/communities/.

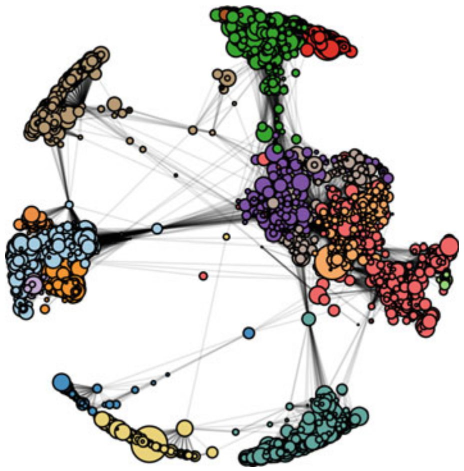
Fig. 8.14 The Facebook network drawn using the Spring layout and different colors to separate the communities found

```
Out[18]: # communities found: 15
```

As can be seen, the 15 communities found automatically are similar to the 10 ego-networks loaded from the dataset (Fig. 8.12). However, some of the 10 ego-networks are subdivided into several communities now. This discrepancy can be due to the fact that the ego-networks are manually annotated based on more properties of the nodes, whereas communities are extracted based only on the graph information.

8.7 Conclusions

In this chapter, we have introduced network analysis and a Python toolbox (NetworkX) that is useful for this analysis. We have shown how network analysis allows us to extract properties from the data that would be hard to discover by other means. Some of these properties are basic concepts in social network analysis, such as centrality measures which return the importance of the nodes in the network or ego-networks which allows us to study the reach of the information a node can transmit or have access to. The different concepts have been practically illustrated by a practical case dealing with a Facebook network. In this practical case, we have resolved several issues, such as finding the most representative members of the network in terms of the most “connected”, the most “circulated”, the “closest”, or the most “accessible” nodes to the others. We have presented useful ways of extracting basic properties of the Facebook network, and studying its ego-networks and communities, as well as comparing them quantitatively and qualitatively. We have also proposed several visualizations of the graph to represent several measures and to emphasize the important nodes with different meanings.

Acknowledgements This chapter was co-written by Laura Igual and Santi Seguí.

References

1. N. Friedkin, *Structural bases of interpersonal influence in groups: A Longitudinal Case Study*. American Sociological Review 58(6):861 1993
2. L. Page, S. Brin, R. Motwani, and T. Winograd, *The PageRank citation ranking: Bringing order to the Web*. 1999
3. V. D. Blondel, J.-L. Guillaume, R. Lambiotte, R. Lefebvre, *Fast unfolding of communities in large networks*. Journal of Statistical Mechanics: Theory and Experiment. 2008(10)