A similar chapter in the book by Chakrabarti [85] also discusses many Web specific issues and has influenced the writing of this chapter. Below, we discuss some further readings related to Web search and mining.

On index compression, Elias coding was introduced by Elias [161] and Golomb coding was introduced by Golomb [202]. Their applications to index compression was studied by several researchers, e.g., Witten et al. [551], Bell et al. [45], Moffat et al. [392], and Williams and Zobel [547]. Wikipedia is a great source of information on this topic as well.

Latent semantic index (LSI) was introduced by Deerwester et al. [125], which uses the singular value decomposition technique (SVD) [203]. Additional information about LSI and/or SVD can be found in [48, 581, 288]. Telcordia Technologies, where LSI was developed, maintains a LSI page at http://lsi.research.telcordia.com/ with more references.

On Web page pre-processing, the focus has been on identifying the main content blocks of each page because a typical Web page contains a large amount of noise, which can adversely affect the search or mining accuracy. Several researchers have attempted the task, e.g., Bar-Yossef et al. [38], Debnath et al. [124], Gibson, et al. [199], Li et al. [324], Lin and Ho [336], Ma et al. [355], Ramaswamy et al. [456], Song et al. [495], Yi et al. [576], Yin and Lee [579], etc.

Although search is probably the biggest application on the Web, little is known about the actual implementation of a search engine except some principal ideas. Sect. 6.8 is largely based on the Google paper by Brin and Page [68], and bits and pieces in various other sources. Over the years, a large number of researchers have studied Web search. More recent studies on various aspects of search can be found in [37, 79, 89, 262, 289, 297, 451, 460, 508, 567, 569, 611].

For metasearch, the combination methods in Sect. 6.9.1 were proposed by Fox and Shaw [184]. Aslam and Montague [28], Montague and Aslam [394], and Nuray and Can [418] provide good descriptions of Borda ranking and Condorcet ranking. In addition to ranking, Meng et al. [378] discussed many other metasearch issues.

On Web spam, Gyongyi and Garcia-Molina gave an excellent taxonomy of different types of spam [214]. The TrustRank algorithm is also due to them [216]. An improvement to TrustRank was proposed by Wu et al. [557]. General link spam detection was studied by Adali et al. [1], Amitay et al. [19], Baeza-Yates et al. [30], Gyongyi and Garcia-Molina [215], Wu and Davison [555], Zhang et al. [604], etc. Content spam detection was studied by Fetterly et al. [176, 177], and Ntoulas et al. [417]. A cloaking detection algorithm is reported in [556].

7 Link Analysis

Early search engines retrieved relevant pages for the user based primarily on the content similarity of the user query and the indexed pages of the search engines. The retrieval and ranking algorithms were simply direct implementation of those from information retrieval. Starting from 1996, it became clear that content similarity alone was no longer sufficient for search due to two reasons. First, the number of Web pages grew rapidly during the middle to late 1990s. Given any query, the number of relevant pages can be huge. For example, given the search query "classification technique", the Google search engine estimates that there are about 10 million relevant pages. This abundance of information causes a major problem for ranking, i.e., how to choose only 30–40 pages and rank them suitably to present to the user. Second, content similarity methods are easily spammed. A page owner can repeat some important words and add many remotely related words in his/her pages to boost the rankings of the pages and/or to make the pages relevant to a large number of possible queries.

Starting from around 1996, researchers in academia and search engine companies began to work on the problem. They resort to hyperlinks. Unlike text documents used in traditional information retrieval, which are often considered independent of one another (i.e., with no explicit relationships or links among them except in citation analysis), Web pages are connected through hyperlinks, which carry important information. Some hyperlinks are used to organize a large amount of information at the same Web site, and thus only point to pages in the same site. Other hyperlinks point to pages in other Web sites. Such out-going hyperlinks often indicate an implicit conveyance of authority to the pages being pointed to. Therefore, those pages that are pointed to by many other pages are likely to contain authoritative or quality information. Such linkages should obviously be used in page evaluation and ranking in search engines.

During the period of 1997-1998, two most influential hyperlink based search algorithms PageRank [68, 422] and HITS [281] were designed. PageRank is the algorithm that powers the successful search engine Google. Both PageRank and HITS were originated from **social network analysis** [540]. They both exploit the hyperlink structure of the Web to rank pages according to their levels of "prestige" or "authority". We will study these

algorithms in this chapter. We should also note that hyperlink-based page evaluation and ranking is not the only method used by search engines. As we discussed in Chap. 6, contents and many other factors are also considered in producing the final ranking presented to the user.

Apart from search ranking, hyperlinks are also useful for finding Web communities. A Web community is a cluster of densely linked pages representing a group of people with a common interest. Beyond explicit hyperlinks on the Web, links in other contexts are useful too, e.g., for discovering communities of named entities (e.g., people and organizations) in free text documents, and for analyzing social phenomena in emails. This chapter will introduce some of the current algorithms.

7.1 Social Network Analysis

Social network is the study of social entities (people in an organization, called **actors**), and their interactions and relationships. The interactions and relationships can be represented with a network or graph, where each vertex (or node) represents an actor and each link represents a relationship. From the network we can study the properties of its structure, and the role, position and prestige of each social actor. We can also find various kinds of sub-graphs, e.g., **communities** formed by groups of actors.

Social network analysis is useful for the Web because the Web is essentially a virtual society, and thus a virtual social network, where each page can be regarded as a social actor and each hyperlink as a relationship. Many of the results from social networks can be adapted and extended for use in the Web context. The ideas from social network analysis are indeed instrumental to the success of Web search engines.

In this section, we introduce two types of social network analysis, **centrality** and **prestige**, which are closely related to hyperlink analysis and search on the Web. Both centrality and prestige are measures of degree of prominence of an actor in a social network. We introduce them below. For a more complete treatment of the topics, please refer to the authoritative text by Wasserman and Faust [540].

7.1.1 Centrality

Important or prominent actors are those that are linked or involved with other actors extensively. In the context of an organization, a person with extensive contacts (links) or communications with many other people in the organization is considered more important than a person with relatively

fewer contacts. The links can also be called **ties**. A **central actor** is one involved in many ties. Fig. 7.1 shows a simple example using an undirected graph. Each node in the social network is an actor and each link indicates that the actors on the two ends of the link communicate with each other. Intuitively, we see that the actor i is the most central actor because he/she can communicate with most other actors.

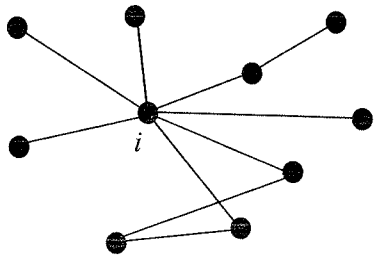

Fig. 7.1. An example of a social network

There are different types of links or involvements between actors. Thus, several types of centrality are defined on undirected and directed graphs. We discuss three popular types below.

Degree Centrality

Central actors are the most active actors that have most links or ties with other actors. Let the total number of actors in the network be n.

Undirected Graph: In an undirected graph, the **degree centrality** of an actor i (denoted by $C_D(i)$) is simply the node degree (the number of edges) of the actor node, denoted by $d(i)$, normalized with the maximum degree, $n-1$.

$$C_D(i) = \frac{d(i)}{n-1}. \tag{1}$$

The value of this measure ranges between 0 and 1 as $n-1$ is the maximum value of $d(i)$.

Directed Graph: In this case, we need to distinguish **in-links** of actor i (links pointing to i), and **out-links** (links pointing out from i). The degree centrality is defined based on only the out-degree (the number of out-links or edges), $d_o(i)$.

$$C_D'(i) = \frac{d_o(i)}{n-1}. \tag{2}$$

Closeness Centrality

This view of centrality is based on the closeness or distance. The basic idea is that an actor x_i is central if it can easily interact with all other actors. That is, its distance to all other actors is short. Thus, we can use the shortest distance to compute this measure. Let the shortest distance from actor i to actor j be $d(i, j)$ (measured as the number of links in a shortest path).

Undirected Graph: The closeness centrality $C_C(i)$ of actor i is defined as

$$C_C(i) = \frac{n-1}{\sum_{j=1}^{n} d(i,j)}. \quad (3)$$

The value of this measure also ranges between 0 and 1 as $n-1$ is the minimum value of the denominator, which is the sum of the shortest distances from i to all other actors. Note that this equation is only meaningful for a connected graph.

Directed Graph: The same equation can be used for a directed graph. The distance computation needs to consider directions of links or edges.

Betweenness Centrality

If two non-adjacent actors j and k want to interact and actor i is on the path between j and k, then i may have some control over their interactions. Betweenness measures this control of i over other pairs of actors. Thus, if i is on the paths of many such interactions, then i is an important actor.

Undirected Graph: Let p_{jk} be the number of shortest paths between actors j and k. The betweenness of an actor i is defined as the number of shortest paths that pass i (denoted by $p_{jk}(i)$, $j \neq i$ and $k \neq i$) normalized by the total number of shortest paths of all pairs of actors not including i:

$$C_B(i) = \sum_{j<k} \frac{p_{jk}(i)}{p_{jk}}. \quad (4)$$

Note that there may be multiple shortest paths between actor j and actor k. Some pass i and some do not. We assume that all paths are equally likely to be used. $C_B(i)$ has a minimum of 0, attained when i falls on no shortest path. Its maximum is $(n-1)(n-2)/2$, which is the number of pairs of actors not including i.

In the network of Fig. 7.2, actor 1 is the most central actor. It lies on all 15 shortest paths linking the other 6 actors. $C_B(1)$ has the maximum value of 15, and $C_B(2) = C_B(3) = C_B(4) = C_B(5) = C_B(6) = C_B(7) = 0$.

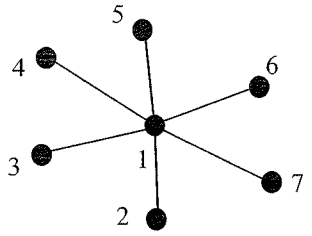

Fig. 7.2. An example of a network illustrating the betweenness centrality

If we are to ensure that the value range is between 0 and 1, we can normalize it with $(n-1)(n-2)/2$, which is the maximum value of $C_B(i)$. The standardized betweenness of actor i is defined as

$$C_B'(i) = \frac{2\sum_{j<k} \frac{p_{jk}(i)}{p_{jk}}}{(n-1)(n-2)}. \quad (5)$$

Unlike the closeness measure, the betweenness can be computed even if the graph is not connected.

Directed Graph: The same equation can be used but must be multiplied by 2 because there are now $(n-1)(n-2)$ pairs considering a path from j to k is different from a path from k to j. Likewise, p_{jk} must consider paths from both directions.

7.1.2 Prestige

Prestige is a more refined measure of prominence of an actor than centrality as we will see below. We need to distinguish between ties sent (out-links) and ties received (in-links). A prestigious actor is defined as one who is object of extensive ties as a recipient. In other words, to compute the prestige of an actor, we only look at the ties (links) directed or pointed to the actor (in-links). Hence, the prestige cannot be computed unless the relation is directional or the graph is directed. The main difference between the concepts of centrality and prestige is that centrality focuses on out-links while prestige focuses on in-links. We define three prestige measures. The third prestige measure (i.e., **rank prestige**) forms the basis of most Web page link analysis algorithms, including PageRank and HITS.

Degree Prestige

Based on the definition of the prestige, it is clear that an actor is prestigious if it receives many in-links or nominations. Thus, the simplest measure of prestige of an actor i (denoted by $P_D(i)$) is its in-degree.

$$P_D(i) = \frac{d_I(i)}{n-1}, \tag{6}$$

where $d_I(i)$ is the in-degree of i (the number of in-links of i) and n is the total number of actors in the network. As in the degree centrality, dividing by $n - 1$ standardizes the prestige value to the range from 0 and 1. The maximum prestige value is 1 when every other actor links to or chooses actor i.

Proximity Prestige

The degree index of prestige of an actor i only considers the actors that are adjacent to i. The proximity prestige generalizes it by considering both the actors directly and indirectly linked to actor i. That is, we consider every actor j that can reach i, i.e., there is a directed path from j to i.

Let I_i be the set of actors that can reach actor i, which is also called the **influence domain** of actor i. The **proximity** is defined as closeness or distance of other actors to i. Let $d(j, i)$ denote the shortest path distance from actor j to actor i. Each link has the unit distance. To compute the proximity prestige, we use the average distance, which is

$$\frac{\sum_{j \in I_i} d(j,i)}{|I_i|}, \tag{7}$$

where $|I_i|$ is the size of the set I_i. If we look at the ratio or proportion of actors who can reach i to the average distance that these actors are from i, we obtain the proximity prestige, which has the value range of [0, 1]:

$$P_P(i) = \frac{|I_i|/(n-1)}{\sum_{j \in I_i} d(j,i) \Big/ |I_i|}, \tag{8}$$

where $|I_i|/(n-1)$ is the proportion of actors that can reach actor i. In one extreme, every actor can reach actor i, which gives $|I_i|/(n-1) = 1$. The denominator is 1 if every actor is adjacent to i. Then, $P_P(i) = 1$. On the other extreme, no actor can reach actor i. Then $|I_i| = 0$, and $P_P(i) = 0$.

Rank Prestige

The above two prestige measures are based on in-degrees and distances. However, an important factor that has not been considered is the **prominence** of individual actors who do the "voting" or "choosing." In the real world, a person i chosen by an important person is more prestigious than chosen by a less important person. For example, a company CEO voting for a person is much more important than a worker voting for the person. If one's circle of influence is full of prestigious actors, then one's own prestige is also high. Thus one's prestige is affected by the ranks or statuses of the involved actors. Based on this intuition, the rank prestige $P_R(i)$ is defined as a linear combination of links that point to i:

$$P_R(i) = A_{1i}P_R(1) + A_{2i}P_R(2) + \ldots + A_{ni}P_R(n), \tag{9}$$

where $A_{ji} = 1$ if j points to i, and 0 otherwise. This equation says that an actor's rank prestige is a function of the ranks of the actors who vote or choose the actor, which makes perfect sense.

Since we have n equations for n actors, we can write them in the matrix notation. We use $\boldsymbol{P}$ to represent the vector that contains all the rank prestige values, i.e., $\boldsymbol{P} = (P_R(1), P_R(2), \ldots, P_R(n))^T$ (T means **matrix transpose**). $\boldsymbol{P}$ is represented as a column vector. We use matrix $\boldsymbol{A}$ (where $A_{ij} = 1$ if i points to j, and 0 otherwise) to represent the adjacency matrix of the network or graph. As a notational convention, we use bold italic letters to represent matrices. We then have

$$\boldsymbol{P} = \boldsymbol{A}^T\boldsymbol{P}. \tag{10}$$

This equation is precisely the characteristic equation used for finding the **eigensystem** of the matrix $\boldsymbol{A}^T$. $\boldsymbol{P}$ is an **eigenvector** of $\boldsymbol{A}^T$.

This equation and the idea behind it turn out to be very useful in Web search. Indeed, the most well known ranking algorithms for Web search, PageRank and HITS, are directly related to this equation. Sect. 7.3 and 7.4 will focus on these two algorithms and describe how to solve the equation to obtain the prestige value of each actor (or each page on the Web).

7.2 Co-Citation and Bibliographic Coupling

Another area of research concerned with links is the **citation analysis** of scholarly publications. A scholarly publication usually cites related prior work to acknowledge the origins of some ideas in the publication and to compare the new proposal with existing work. Citation analysis is an area

of bibliometric research, which studies citations to establish the relationships between authors and their work.

When a publication (also called a paper) cites another publication, a relationship is established between the publications. Citation analysis uses these relationships (links) to perform various types of analysis. A citation can represent many types of links, such as links between authors, publications, journals and conferences, and fields, or even between countries. We will discuss two specific types of citation analysis, **co-citation** and **bibliographic coupling**. The HITS algorithm of Sect. 7.4 is related to these two types of analysis.

7.2.1 Co-Citation

Co-citation is used to measure the similarity of two documents. If papers i and j are both cited by paper k, then they may be said to be related in some sense to one another, even they do not directly cite each other. Figure 7.3 shows that papers i and j are co-cited by paper k. If papers i and j are cited together by many papers, it means that i and j have a strong relationship or similarity. The more papers they are cited by, the stronger their relationship is.

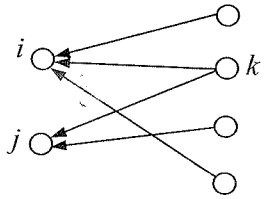

Fig. 7.3. Paper i and paper j are co-cited by paper k

Let $\boldsymbol{L}$ be the citation matrix. Each cell of the matrix is defined as follows: $L_{ij} = 1$ if paper i cites paper j, and 0 otherwise. **Co-citation** (denoted by C_{ij}) is a similarity measure defined as the number of papers that co-cite i and j, and is computed with

$$C_{ij} = \sum_{k=1}^{n} L_{ki} L_{kj}, \tag{11}$$

where n is the total number of papers. C_{ii} is naturally the number of papers that cite i. A square matrix $\boldsymbol{C}$ can be formed with C_{ij}, and it is called the **co-citation matrix**. Co-citation is symmetric, $C_{ij} = C_{ji}$, and is commonly used as a similarity measure of two papers in clustering to group papers of similar topics together.

7.2.2 Bibliographic Coupling

Bibliographic coupling operates on a similar principle, but in a way it is the mirror image of co-citation. Bibliographic coupling links papers that cite the same articles so that if papers i and j both cite paper k, they may be said to be related, even though they do not directly cite each other. The more papers they both cite, the stronger their similarity is. Figure 7.4 shows both papers i and j citing (referencing) paper k.

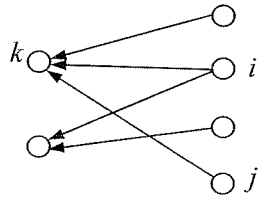

Fig. 7.4. Both paper i and paper j cite paper k

We use B_{ij} to represent the number of papers that are cited by both papers i and j:

$$B_{ij} = \sum_{k=1}^{n} L_{ik} L_{jk}. \tag{12}$$

B_{ii} is naturally the number of references (in the reference list) of paper i. A square matrix $\boldsymbol{B}$ can be formed with B_{ij}, and it is called the **bibliographic coupling matrix**. Bibliographic coupling is also symmetric and is regarded as a similarity measure of two papers in clustering.

We will see later that two important types of pages on the Web, **hubs** and **authorities**, found by the HITS algorithm are directly related to co-citation and bibliographic coupling matrices.

7.3 PageRank

The year 1998 was an important year for Web link analysis and Web search. Both the PageRank and the HITS algorithms were reported in that year. HITS was presented by Jon Kleinberg in January, 1998 at the *Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. PageRank was presented by Sergey Brin and Larry Page at the *Seventh International World Wide Web Conference* (*WWW7*) in April, 1998. Based on the algorithm, they built the search engine Google. The main ideas of PageRank and HITS are really quite similar. However, it is their dissimilarity that

made a huge difference as we will see later. Since that year, PageRank has emerged as the dominant link analysis model for Web search, partly due to its query-independent evaluation of Web pages and its ability to combat spamming, and partly due to Google's business success. In this section, we focus on PageRank. In the next section, we discuss HITS. A detailed study of these algorithms can also be found in [304].

PageRank relies on the democratic nature of the Web by using its vast link structure as an indicator of an individual page's quality. In essence, PageRank interprets a hyperlink from page x to page y as a vote, by page x, for page y. However, PageRank looks at more than just the sheer number of votes, or links that a page receives. It also analyzes the page that casts the vote. Votes casted by pages that are themselves "important" weigh more heavily and help to make other pages more "important." This is exactly the idea of **rank prestige** in social networks (see Sect. 7.1.2).

7.3.1 PageRank Algorithm

PageRank is a static ranking of Web pages in the sense that a PageRank value is computed for each page off-line and it does not depend on search queries. Since PageRank is based on the measure of prestige in social networks, the PageRank value of each page can be regarded as its prestige. We now derive the PageRank formula. Let us first state some main concepts again in the Web context.

In-links of page i: These are the hyperlinks that point to page i from other pages. Usually, hyperlinks from the same site are not considered.

Out-links of page i: These are the hyperlinks that point out to other pages from page i. Usually, links to pages of the same site are not considered.

From the perspective of prestige, we use the following to derive the PageRank algorithm.

1. A hyperlink from a page pointing to another page is an implicit conveyance of authority to the target page. Thus, the more in-links that a page i receives, the more prestige the page i has.
2. Pages that point to page i also have their own prestige scores. A page with a higher prestige score pointing to i is more important than a page with a lower prestige score pointing to i. In other words, a page is important if it is pointed to by other important pages.

According to rank prestige in social networks, the importance of page i (i's PageRank score) is determined by summing up the PageRank scores of all pages that point to i. Since a page may point to many other pages, its prestige score should be shared among all the pages that it points to. Notice the difference from rank prestige, where the prestige score is not shared.

To formulate the above ideas, we treat the Web as a directed graph $G = (V, E)$, where V is the set of vertices or nodes, i.e., the set of all pages, and E is the set of directed edges in the graph, i.e., hyperlinks. Let the total number of pages on the Web be n (i.e., $n = |V|$). The PageRank score of the page i (denoted by $P(i)$) is defined by:

$$P(i) = \sum_{(j,i)\in E} \frac{P(j)}{O_j}, \tag{13}$$

where O_j is the number of out-links of page j. Mathematically, we have a system of n linear equations (13) with n unknowns. We can use a matrix to represent all the equations. Let $\boldsymbol{P}$ be a n-dimensional column vector of PageRank values, i.e.,

$$\boldsymbol{P} = (P(1), P(2), \ldots, P(n))^T.$$

Let $\boldsymbol{A}$ be the adjacency matrix of our graph with

$$A_{ij} = \begin{cases} \frac{1}{O_i} & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

We can write the system of n equations with (similar to Equation 10)

$$\boldsymbol{P} = \boldsymbol{A}^T \boldsymbol{P}. \tag{15}$$

This is the characteristic equation of the **eigensystem**, where the solution to $\boldsymbol{P}$ is an **eigenvector** with the corresponding **eigenvalue** of 1. Since this is a circular definition, an iterative algorithm is used to solve it. It turns out that if *some conditions* are satisfied (which will be described shortly), 1 is the largest **eigenvalue** and the PageRank vector $\boldsymbol{P}$ is the **principal eigenvector**. A well known mathematical technique called **power iteration** can be used to find $\boldsymbol{P}$.

However, the problem is that Equation (15) does not quite suffice because the Web graph does not meet the conditions. To introduce these conditions and the enhanced equation, let us derive the same Equation (15) based on the **Markov chain** [207].

In the Markov chain model, each Web page or node in the Web graph is regarded as a state. A hyperlink is a transition, which leads from one state to another state with a probability. Thus, this framework models Web surfing as a stochastic process. It models a Web surfer randomly surfing the Web as a state transition in the Markov chain. Recall that we used O_i to

denote the number of out-links of a node i. Each transition probability is $1/O_i$ if we assume the Web surfer will click the hyperlinks in the page i uniformly at random, the "back" button on the browser is not used and the surfer does not type in an URL. Let $\boldsymbol{A}$ be the state transition probability matrix, a square matrix of the following format,

$$\boldsymbol{A} = \begin{pmatrix} A_{11} & A_{12} & . & . & . & A_{1n} \\ A_{21} & A_{22} & . & . & . & A_{2n} \\ . & . & & & & . \\ . & . & & & & . \\ . & . & & & & . \\ A_{n1} & A_{n2} & . & . & . & A_{nn} \end{pmatrix}$$

A_{ij} represents the transition probability that the surfer in state i (page i) will move to state j (page j). A_{ij} is defined exactly as in Equation (14).

Given an **initial probability distribution** vector that a surfer is at each state (or page) $p_0 = (p_0(1), p_0(2), \ldots, p_0(n))^T$ (a column vector) and an $n \times n$ **transition probability matrix** $\boldsymbol{A}$, we have

$$\sum_{i=1}^{n} p_0(i) = 1 \tag{16}$$

$$\sum_{j=1}^{n} A_{ij} = 1. \tag{17}$$

Equation (17) is not quite true for some Web pages because they have no out-links. If the matrix $\boldsymbol{A}$ satisfies Equation (17), we say that $\boldsymbol{A}$ is the **stochastic matrix** of a Markov chain. Let us assume $\boldsymbol{A}$ is a stochastic matrix for the time being and deal with it not being that later.

In a Markov chain, a question of common interest is: Given the initial probability distribution $\boldsymbol{p}_0$ at the beginning, what is the probability that m steps/transitions later that the Markov chain will be at each state j? We can determine the probability that the system (or the **random surfer**) is in state j after 1 step (1 state transition) by using the following reasoning:

$$p_1(j) = \sum_{i=1}^{n} A_{ij}(1) p_0(i), \tag{18}$$

where $A_{ij}(1)$ is the probability of going from i to j after 1 transition, and $A_{ij}(1) = A_{ij}$. We can write it with a matrix:

$$\boldsymbol{p}_1 = \boldsymbol{A}^T \boldsymbol{p}_0 . \tag{19}$$

In general, the probability distribution after k steps/transitions is:

$$\boldsymbol{p}_k = \boldsymbol{A}^T \boldsymbol{p}_{k-1} . \tag{20}$$

Equation (20) looks very similar to Equation (15). We are getting there.

By the Ergodic Theorem of Markov chains [207], a finite Markov chain defined by the **stochastic transition matrix** $\boldsymbol{A}$ has a unique **stationary probability distribution** if $\boldsymbol{A}$ is **irreducible** and **aperiodic**. These mathematical terms will be defined as we go along.

The stationary probability distribution means that after a series of transitions $\boldsymbol{p}_k$ will converge to a steady-state probability vector $\boldsymbol{\pi}$ regardless of the choice of the initial probability vector $\boldsymbol{p}_0$, i.e.,

$$\lim_{k \to \infty} \boldsymbol{p}_k = \boldsymbol{\pi} . \tag{21}$$

When we reach the steady-state, we have $\boldsymbol{p}_k = \boldsymbol{p}_{k+1} = \boldsymbol{\pi}$, and thus $\boldsymbol{\pi} = \boldsymbol{A}^T \boldsymbol{\pi}$. $\boldsymbol{\pi}$ is the **principal eigenvector** of $\boldsymbol{A}^T$ with **eigenvalue** of 1. In PageRank, $\boldsymbol{\pi}$ is used as the PageRank vector $\boldsymbol{P}$. Thus, we again obtain Equation (15), which is re-produced here as Equation (22):

$$\boldsymbol{P} = \boldsymbol{A}^T \boldsymbol{P} . \tag{22}$$

Using the stationary probability distribution $\boldsymbol{\pi}$ as the PageRank vector is reasonable and quite intuitive because it reflects the long-run probabilities that a random surfer will visit the pages. A page has a high prestige if the probability of visiting it is high.

Now let us come back to the real Web context and see whether the above conditions are satisfied, i.e., whether $\boldsymbol{A}$ is a stochastic matrix and whether it is irreducible and aperiodic. In fact, none of them is satisfied. Hence, we need to extend the ideal-case Equation (22) to produce the "actual PageRank model". Let us look at each condition below.

First of all, $\boldsymbol{A}$ is not a **stochastic (transition) matrix**. A stochastic matrix is the transition matrix for a finite Markov chain whose entries in each row are non-negative real numbers and sum to 1 (i.e., Equation 17). This requires that every Web page must have at least one out-link. This is not true on the Web because many pages have no out-links, which are reflected in transition matrix $\boldsymbol{A}$ by some rows of complete 0's. Such pages are called the **dangling pages** (nodes).

Example 1: Figure 7.5 shows an example of a hyperlink graph.

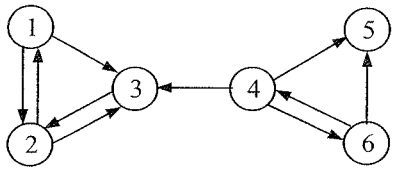

Fig. 7.5. An example of a hyperlink graph

If we assume that the Web surfer will click the hyperlinks in a page uniformly at random, we have the following transition probability matrix:

$$A = \begin{pmatrix} 0 & 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 0 & 1/3 & 1/3 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 \end{pmatrix}. \tag{23}$$

For example $A_{12} = A_{13} = 1/2$ because node 1 has two out-links. We can see that A is not a stochstic matrix because the fifth row is all 0's, i.e., page 5 is a dangling page. ■

We can fix this problem in several ways in order to convert A to a stochastic transition matrix. We describe only two ways here:

1. Remove those pages with no out-links from the system during the PageRank computation as these pages do not affect the ranking of any other page directly. Out-links from other pages pointing to these pages are also removed. After PageRanks are computed, these pages and hyperlinks pointing to them can be added in. Their PageRanks are easy to calculate based on Equation (22). Note that the transition probabilities of those pages with removed links will be slightly affected but not significantly. This method is suggested in [68].
2. Add a complete set of outgoing links from each such page i to all the pages on the Web. Thus the transition probability of going from i to every page is $1/n$ assuming uniform probability distribution. That is, we replace each row containing all 0's with $\boldsymbol{e}/n$, where $\boldsymbol{e}$ is n-dimensional vector of all 1's.

If we use the second method to make A a stochastic matrix by adding a link from page 5 to every page, we obtain

$$\overline{A} = \begin{pmatrix} 0 & 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 0 & 1/3 & 1/3 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 \end{pmatrix}. \tag{24}$$

Below, we assume that either one of the above is done to make A a stochastic matrix.

Second, A is not **irreducible**. Irreducible means that the Web graph G is strongly connected.

Definition (strongly connected): A directed graph $G = (V, E)$ is **strongly connected** if and only if, for each pair of nodes $u, v \in V$, there is a path from u to v.

A general Web graph represented by A is not irreducible because for some pair of nodes u and v, there is no path from u to v. For example, in Fig. 7.5, there is no directed path from node 3 to node 4. The adjustment in Equation (24) is not enough to ensure irreducibility. That is, in $\overline{A}$, there is still no directed path from node 3 to node 4. This problem and the next problem can be dealt with using a single strategy (to be described shortly).

Finally, A is not **aperiodic**. A state i in a Markov chain being periodic means that there exists a directed cycle that the chain has to traverse.

Definition (aperiodic): A state i is **periodic** with period $k > 1$ if k is the smallest number such that all paths leading from state i back to state i have a length that is a multiple of k. If a state is not periodic (i.e., $k = 1$), it is **aperiodic**. A Markov chain is **aperiodic** if all states are aperiodic.

Example 2: Figure 7.6 shows a periodic Markov chain with $k = 3$. The transition matrix is given on the left. Each state in this chain has a period of 3. For example, if we start from state 1, to come back to state 1 the only path is 1-2-3-1 for some number of times, say h. Thus any return to state 1 will take $3h$ transitions. In the Web, there could be many such cases. ■

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

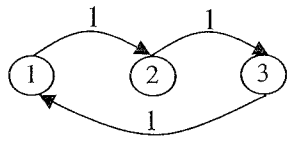

Fig. 7.6. A periodic Markov chain with $k = 3$.

It is easy to deal with the above two problems with a single strategy.

- We add a link from each page to every page and give each link a small transition probability controlled by a parameter d.

The augmented transition matrix becomes **irreducible** because it is clearly strongly connected. It is also **aperiodic** because the situation in Fig. 7.6 no longer exists as we now have paths of all possible lengths from state i back to state i. That is, the random surfer does not have to traverse a fixed cycle for any state. After this augmentation, we obtain an improved PageRank model. In this model, at a page, the random surfer has two options:

1. With probability d, he randomly chooses an out-link to follow.
2. With probability $1-d$, he jumps to a random page without a link.

Equation (25) gives the improved model,

$$\boldsymbol{P} = \left((1-d)\frac{\boldsymbol{E}}{n} + d\boldsymbol{A}^T \right) \boldsymbol{P} \tag{25}$$

where $\boldsymbol{E}$ is $\boldsymbol{e}\boldsymbol{e}^T$ ($\boldsymbol{e}$ is a column vector of all 1's) and thus $\boldsymbol{E}$ is a $n \times n$ square matrix of all 1's. $1/n$ is the probability of jumping to a particular page. n is the total number of nodes in the Web graph. Note that Equation (25) assumes that $\boldsymbol{A}$ has already been made a stochastic matrix.

Example 3: If we follow our example in Fig. 7.5 and Equation (24) (we use $\overline{\boldsymbol{A}}$ for $\boldsymbol{A}$ here), the augmented transition matrix is

$$(1-d)\frac{\boldsymbol{E}}{n} + d\boldsymbol{A}^T = \begin{pmatrix} 1/60 & 7/15 & 1/60 & 1/60 & 1/6 & 1/60 \\ 7/15 & 1/60 & 11/12 & 1/60 & 1/6 & 1/60 \\ 7/15 & 7/15 & 1/60 & 19/60 & 1/6 & 1/60 \\ 1/60 & 1/60 & 1/60 & 1/60 & 1/6 & 7/15 \\ 1/60 & 1/60 & 1/60 & 19/60 & 1/6 & 7/15 \\ 1/60 & 1/60 & 1/60 & 19/60 & 1/6 & 1/60 \end{pmatrix} \tag{26}$$

■

$(1-d)E/n + dA^T$ is a **stochastic matrix** (but transposed). It is also **irreducible** and **aperiodic** as we discussed above. Here we use $d = 0.9$.

If we scale Equation (25) so that $\boldsymbol{e}^T\boldsymbol{P} = n$, we obtain

$$\boldsymbol{P} = (1-d)\boldsymbol{e} + d\boldsymbol{A}^T\boldsymbol{P}. \tag{27}$$

Before scaling, we have $\boldsymbol{e}^T\boldsymbol{P} = 1$ (i.e., $P(1) + P(2) + \ldots + P(n) = 1$ if we recall that $\boldsymbol{P}$ is the stationary probability vector π of the Markov chain). The scaling is equivalent to multiplying n on both sides of Equation (25).

This gives us the PageRank formula for each page i as follows:

$$P(i) = (1-d) + d\sum_{j=1}^{n} A_{ji}P(j), \tag{28}$$

which is equivalent to the formula given in the PageRank papers [68, 422]:

$$P(i) = (1-d) + d\sum_{(j,i)\in E} \frac{P(j)}{O_j}. \tag{29}$$

The parameter d is called the **damping factor** which can be set to between 0 and 1. $d = 0.85$ is used in [68, 422].

The computation of PageRank values of the Web pages can be done using the well known **power iteration method** [203], which produces the principal eigenvector with the eigenvalue of 1. The algorithm is simple, and is given in Fig. 7.7. One can start with any initial assignments of PageRank values. The iteration ends when the PageRank values do not change much or converge. In Fig. 7.7, the iteration ends after the 1-norm of the residual vector is less than a pre-specified threshold ε. Note that the 1-norm for a vector is simply the sum of all the components.

PageRank-Iterate(G)
  $\boldsymbol{P}_0 \leftarrow \boldsymbol{e}/n$
  $k \leftarrow 1$
  repeat
    $\boldsymbol{P}_k \leftarrow (1-d)\boldsymbol{e} + d\boldsymbol{A}^T\boldsymbol{P}_{k-1}$;
    $k \leftarrow k+1$;
  until $||\boldsymbol{P}_k - \boldsymbol{P}_{k-1}||_1 < \varepsilon$
  return $\boldsymbol{P}_k$

Fig. 7.7. The power iteration method for PageRank

Since we are only interested in the ranking of the pages, the actual convergence may not be necessary. Thus, fewer iterations are needed. In [68], it is reported that on a database of 322 million links the algorithm converges to an acceptable tolerance in roughly 52 iterations.

7.3.2 Strengths and Weaknesses of PageRank

The main advantage of PageRank is its ability to fight spam. A page is important if the pages pointing to it are important. Since it is not easy for Web page owner to add in-links into his/her page from other important pages, it is thus not easy to influence PageRank. Nevertheless, there are

reported ways to influence PageRank. Recognizing and fighting spam is an important issue in Web search.

Another major advantage of PageRank is that it is a global measure and is query independent. That is, the PageRank values of all the pages on the Web are computed and saved off-line rather than at the query time. At the query time, only a lookup is needed to find the value to be integrated with other strategies to rank the pages. It is thus very efficient at query time. Both these two advantages contributed greatly to Google's success.

The main criticism is also the query-independence nature of PageRank. It could not distinguish between pages that are authoritative in general and pages that are authoritative on the query topic. Google may have other ways to deal with the problem, which we do not know due to the proprietary nature of Google. Another criticism is that PageRank does not consider time. Let us give some explanation to this.

7.3.3 Timed PageRank

The Web is a dynamic environment, and it changes constantly. Quality pages in the past may not be quality pages now or in the future. Thus, search has a temporal dimension. An algorithm called **TimedPageRank** given in [326, 585] adds the temporal dimension to PageRank. The motivations are:

1. Users are often interested in the latest information. Apart from pages that contain well-established facts and classics which do not change significantly over time, most contents on the Web change constantly. New pages or contents are added, and ideally, outdated contents and pages are deleted. However, in practice many outdated pages and links are not deleted. This causes problems for Web search because such outdated pages may still be ranked very high.
2. PageRank favors pages that have many in-links. To some extent, we can say that it favors older pages because they have existed on the Web for a long time and thus have accumulated many in-links. Then the problem is that new pages which are of high quality and also give the up-to-date information will not be assigned high scores and consequently will not be ranked high because they have fewer or no in-links. It is thus difficult for users to find the latest information on the Web based on PageRank.

The idea of TimedPageRank is simple. Instead of using a constant damping factor d as the parameter in PageRank, TimedPageRank uses a function of time $f(t)$ ($0 \leq f(t) \leq 1$), where t is the difference between the current time and the time when the page was last updated. $f(t)$ returns a probability that the Web surfer will follow an actual link on the page. $1-f(t)$ returns the probability that the surfer will jump to a random page. Thus, at a particular page i, the Web surfer has two options:

1. With probability $f(t_i)$, he randomly chooses an out-going link to follow.
2. With probability $1-f(t_i)$, he jumps to a random page without a link.

The intuition here is that if the page was last updated (or created) a long time ago, the pages that it cites (points to) are even older and are probably out of date. Then the $1-f(t)$ value for such a page should be large, which means that the surfer will have a high probability of jumping to a random page. If a page is new, then its $1-f(t)$ value should be small, which means that the surfer will have a high probability to follow an out-link of the page and a small probability of jumping to a random page.

For a complete new page in a Web site, which does not have any in-links at all, the method given in [326] uses the average TimedPageRank value of the past pages in the Web site.

Finally, we note again that the link-based ranking is not the only strategy used in a search engine. Many other information retrieval methods, heuristics and empirical parameters are also employed. However, their details are not published. We also note that PageRank is not the only link-based static and global ranking algorithm. All major search engines, such as *Yahoo*! and MSN, have their own algorithms but are unpublished.

7.4 HITS

HITS stands for **Hypertext Induced Topic Search** [281]. Unlike PageRank which is a static ranking algorithm, HITS is search query dependent. When the user issues a search query, HITS first expands the list of relevant pages returned by a search engine and then produces two rankings of the expanded set of pages, **authority ranking** and **hub ranking**.

An **authority** is a page with many in-links. The idea is that the page may have good or authoritative content on some topic and thus many people trust it and link to it. A **hub** is a page with many out-links. The page serves as an organizer of the information on a particular topic and points to many good authority pages on the topic. When a user comes to this hub page, he/she will find many useful links which take him/her to good content pages on the topic. Figure 7.8 shows an authority page and a hub page.

The key idea of HITS is that a good hub points to many good authorities and a good authority is pointed to by many good hubs. Thus, authorities and hubs have a **mutual reinforcement** relationship. Figure 7.9 shows a

set of densely linked authorities and hubs (a **bipartite sub-graph**).

Below, we first present the HITS algorithm, and also make a connection between HITS and co-citation and bibliographic coupling in bibliometric research. We then discuss the strengths and weaknesses of HITS, and describe some possible ways to deal with its weaknesses.

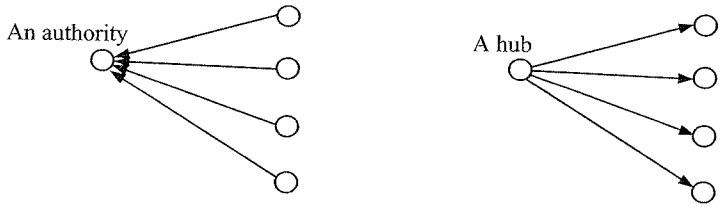

Fig. 7.8. An authority page and a hub page

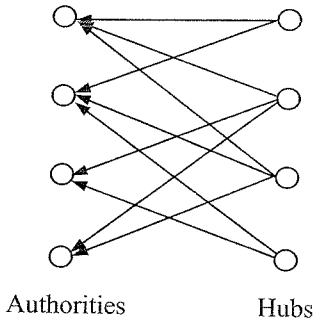

Fig. 7.9. A densely linked set of authorities and hubs

7.4.1 HITS Algorithm

Before describing the HITS algorithm, let us first describe how HITS collects pages to be ranked. Given a broad search query, q, HITS collects a set of pages as follows:

1. It sends the query q to a search engine system. It then collects t ($t = 200$ is used in the HITS paper) highest ranked pages, which assume to be highly relevant to the search query. This set is called the **root** set W.
2. It then grows W by including any page pointed to by a page in W and any page that points to a page in W. This gives a larger set called S. However, this set can be very large. The algorithm restricts its size by allowing each page in W to bring at most k pages ($k = 50$ is used in the HITS paper) pointing to it into S. The set S is called the **base set**.

HITS then works on the pages in S, and assigns every page in S an **authority score** and a **hub score**. Let the number of pages to be studied be n. We again use $G = (V, E)$ to denote the (directed) link graph of S. V is the set of pages (or nodes) and E is the set of directed edges (or links). We use $\boldsymbol{L}$ to denote the adjacency matrix of the graph.

$$L_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{30}$$

Let the authority score of the page i be $a(i)$, and the hub score of page i be $h(i)$. The mutual reinforcing relationship of the two scores is represented as follows:

$$a(i) = \sum_{(j,i)\in E} h(j) \tag{31}$$

$$h(i) = \sum_{(i,j)\in E} a(j) \tag{32}$$

Writing them in the matrix form, we use $\boldsymbol{a}$ to denote the column vector with all the authority scores, $\boldsymbol{a} = (a(1), a(2), \ldots, a(n))^T$, and use $\boldsymbol{h}$ to denote the column vector with all the authority scores, $\boldsymbol{h} = (h(1), h(2), \ldots, h(n))^T$,

$$\boldsymbol{a} = \boldsymbol{L}^T\boldsymbol{h} \tag{33}$$

$$\boldsymbol{h} = \boldsymbol{L}\boldsymbol{a} \tag{34}$$

The computation of authority scores and hub scores is basically the same as the computation of the PageRank scores using the power iteration method. If we use $\boldsymbol{a}_k$ and $\boldsymbol{h}_k$ to denote authority and hub scores at the kth iteration, the iterative processes for generating the final solutions are

$$\boldsymbol{a}_k = \boldsymbol{L}^T\boldsymbol{L}\boldsymbol{a}_{k-1} \tag{35}$$

$$\boldsymbol{h}_k = \boldsymbol{L}\boldsymbol{L}^T\boldsymbol{h}_{k-1} \tag{36}$$

starting with

$$\boldsymbol{a}_0 = \boldsymbol{h}_0 = (1, 1, \ldots, 1). \tag{37}$$

Note that Equation (35) (or Equation 36) does not use the hub (or authority) vector due to substitutions of Equation (33) and Equation (34).

After each iteration, the values are also normalized (to keep them small) so that

HITS-Iterate(G)
  $\boldsymbol{a}_0 \leftarrow \boldsymbol{h}_0 \leftarrow (1, 1, \ldots, 1)$;
  $k \leftarrow 1$
  Repeat
    $\boldsymbol{a}_k \leftarrow \boldsymbol{L}^T \boldsymbol{L} \boldsymbol{a}_{k-1}$;
    $\boldsymbol{h}_k \leftarrow \boldsymbol{L}\boldsymbol{L}^T \boldsymbol{h}_{k-1}$;
    $\boldsymbol{a}_k \leftarrow \boldsymbol{a}_k / ||\boldsymbol{a}_k||_1$; // normalization
    $\boldsymbol{h}_k \leftarrow \boldsymbol{h}_k / ||\boldsymbol{h}_k||_1$; // normalization
    $k \leftarrow k + 1$;
  until $||\boldsymbol{a}_k - \boldsymbol{a}_{k-1}||_1 < \varepsilon_a$ and $||\boldsymbol{h}_k - \boldsymbol{h}_{k-1}||_1 < \varepsilon_h$;
  return $\boldsymbol{a}_k$ and $\boldsymbol{h}_k$

Fig. 7.10. The HITS algorithm based on power iteration

$$\sum_{i=1}^{n} a(i) = 1 \tag{38}$$

$$\sum_{i=1}^{n} h(i) = 1 \tag{39}$$

The power iteration algorithm for HITS is given in Fig. 7.10. The iteration ends after the 1-norms of the residual vectors are less than some thresholds ε_a and ε_h. Hence, the algorithm finds the principal eigenvectors at "equilibrium" as in PageRank. The pages with large authority and hub scores are better authorities and hubs respectively. HITS will select a few top ranked pages as authorities and hubs, and return them to the user.

Although HITS will always converge, there is a problem with uniqueness of limiting (converged) authority and hub vectors. It is shown that for certain types of graphs, different initializations to the power method produce different final authority and hub vectors. Some results can be inconsistent or wrong. Farahat et al. [171] gave several examples. The heart of the problem is that there are repeated dominant (principal) eigenvalues (several eigenvalues are the same and are dominant eigenvalues), which are caused by the problem that $\boldsymbol{L}^T\boldsymbol{L}$ (respectively $\boldsymbol{L}\boldsymbol{L}^T$) is reducible [303]. The first PageRank solution (Equation 22) has the same problem. However, the PageRank inventors found a way to get around the problem. A modification similar to PageRank may be applied to HITS.

7.4.2 Finding Other Eigenvectors

The HITS algorithm given in Fig. 7.10 finds the principal eigenvectors, which in a sense represent the most densely connected authorities and hubs in the graph G defined by a query. However, in some cases, we may also be interested in finding several densely linked collections of hubs and authorities among the same base set of pages. Each of such collections could potentially be relevant to the query topic, but they could be well-separated from one another in the graph G for a variety of reasons. For example,

1. The query string may be ambiguous with several very different meanings, e.g., "jaguar", which could be a cat or a car.
2. The query string may represent a topic that may arise as a term in the multiple communities, e.g. "classification".
3. The query string may refer to a highly polarized issue, involving groups that are not likely to link to one another, e.g. "abortion".

In each of these examples, the relevant pages can be naturally grouped into several clusters, also called **communities**. In general, the top ranked authorities and hubs represent the major cluster (or **community**). The smaller clusters (or communities), which are also represented by bipartite subgraphs as that in Fig. 7.9, can be found by computing non-principal eigenvectors. Non-principal eigenvectors are calculated in a similar way to power iteration using methods such as **orthogonal iteration** and **QR iteration**. We will not discuss the details of these methods. Interested readers can refer to the book by Golub and Van Loan [203].

7.4.3 Relationships with Co-Citation and Bibliographic Coupling

Authority pages and hub pages have their matches in the bibliometric citation context. An authority page is like an influential research paper (publication) which is cited by many subsequent papers. A hub page is like a survey paper which cites many other papers (including those influential papers). It is no surprise that there is a connection between authority and hub, and co-citation and bibliographic coupling.

Recall that co-citation of pages i and j, denoted by C_{ij}, is computed as

$$C_{ij} = \sum_{k=1}^{n} L_{ki} L_{kj} = (\boldsymbol{L}^T \boldsymbol{L})_{ij}. \tag{40}$$

This shows that the authority matrix $(\boldsymbol{L}^T\boldsymbol{L})$ of HITS is in fact the co-citation matrix $\boldsymbol{C}$ in the Web context. Likewise, recall that bibliographic

coupling of two pages i and j, denoted by B_{ij}, is computed as

$$B_{ij} = \sum_{k=1}^{n} L_{ik} L_{jk} = (\boldsymbol{L}\boldsymbol{L}^T)_{ij}, \tag{41}$$

which shows that the hub matrix $(\boldsymbol{L}\boldsymbol{L}^T)$ of HITS is the bibliographic coupling matrix $\boldsymbol{B}$ in the Web context.

7.4.4 Strengths and Weaknesses of HITS

The main strength of HITS [281] is its ability to rank pages according to the query topic, which may be able to provide more relevant authority and hub pages. The ranking may also be combined with information retrieval based rankings. However, HITS has several disadvantages.

- First of all, it does not have the anti-spam capability of PageRank. It is quite easy to influence HITS by adding out-links from one's own page to point to many good authorities. This boosts the hub score of the page. Because hub and authority scores are interdependent, it in turn also increases the authority score of the page.
- Another problem of HITS is topic drift. In expanding the root set, it can easily collect many pages (including authority pages and hub pages) which have nothing to do the search topic because out-links of a page may not point to pages that are relevant to the topic and in-links to pages in the root set may be irrelevant as well because people put hyperlinks for all kinds of reasons, including spamming.
- The query time evaluation is also a major drawback. Getting the root set, expanding it and then performing eigenvector computation are all time consuming operations.

Over the years, many researchers tried to deal with these problems. We briefly discuss some of them below.

It was reported by several researchers in [52, 310, 405] that small changes to the Web graph topology can significantly change the final authority and hub vectors. Minor perturbations have little effect on PageRank, which is more stable than HITS. This is essentially due to the random jump step of PageRank. Ng et al. [405] proposed a method by introducing the same random jump step to HITS (by jumping to the base set uniformly at random with probability d), and showed that it could improve the stability of HITS significantly. Lempel and Moran [310] proposed SALSA, *a stochastic algorithm for link structure analysis*. SALSA combines some features of both PageRank and HITS to improve the authority and hub computation. It casts the problem as two Markov chains, an authority Markov chain and a hub Markov chain. SALSA is less susceptible to spam since the coupling between hub and authority scores is much less strict.

Bharat and Henzinger [52] proposed a simple method to fight two site nepotistic links. That means that a set of pages on one host points to a single page on a second host. This drives up the hub scores of the pages on the first host and the authority score of the page on the second host. A similar thing can be done for hubs. These links may be authored by the same person and thus are regarded as "nepotistic" links to drive up the ranking of the target pages. [52] suggests weighting the links to deal with this problem. That is, if there are k edges from documents on a first host to a single document on a second host we give each edge an **authority weight** of $1/k$. If there are l edges from a single page on a first host to a set of pages on a second host, we give each edge a **hub weight** of $1/l$. These weights are used in the authority and hub computation. There are much more sophisticated spam techniques now involving more than two sites.

Regarding the topic drifting of HITS, existing fixes are mainly based on content similarity comparison during the expansion of the root set. In [88], if an expanded page is too different from the pages in the root set in terms of content similarity (based on cosine similarity), it is discarded. The remaining links are also weighted according to similarity. [88] proposes a method that uses the similarity between the anchor text of a link and the search topic to weight the link (instead of giving each link 1 as in HITS). [84] goes further to segment the page based on the DOM (**Document Object Model**) tree structure to identify the blocks or subtrees that are more related to the query topic instead of regarding the whole page as relevant to the search query. This is a good way to deal with multi-topic pages, which are abundant on the Web. A recent work on this is block-based link analysis [78], which segments each Web page into different blocks. Each block is given a different importance value according to its location in the page and other information. The importance value is then used to weight the links in the HITS (and also PageRank) computation. This will reduce the impact of unimportant links, which usually cause topic drifting and may even be a link spam.

7.5 Community Discovery

Intuitively, a community is simply a group of entities (e.g., people or organizations) that shares a common interest or is involved in an activity or event. In Sect. 7.4.2, we showed that the HITS algorithm can be used to find communities. The communities are represented by dense bipartite subgraphs. We now describe several other community finding algorithms.

Apart from the Web, communities also exist in emails and text documents. This section describes two community finding algorithms for the Web, one community finding algorithm for emails, and one community finding algorithm for text documents.

There are many reasons for discovering communities. For example, in the context of the Web, Kumar et al. [293] listed three reasons:

1. Communities provide valuable and possibly the most reliable, timely, and up-to-date information resources for a user interested in them.
2. They represent the sociology of the Web: studying them gives insights into the evolution of the Web.
3. They enable target advertising at a very precise level.

7.5.1 Problem Definition

Definition (community): Given a finite set of **entities** $S = \{s_1, s_2, \ldots, s_n\}$ of the same **type**, a **community** is a pair $C = (T, G)$, where T is the **community theme** and $G \subseteq S$ is the set of all entities in S that shares the theme T. If $s_i \in G$, s_i is said to be a **member** of the community C.

Some remarks about this definition are in order:

- A theme defines a community. That is, given a theme T, the set of members of the community is uniquely determined. Thus, two communities are equal if they have the same theme.
- A theme can be defined arbitrarily. For example, it can be an event (e.g., a sport event or a scandal) or a concept (e.g., Web mining).
- An entity s_i in S can be in any number of communities. That is, communities may overlap, or multiple communities may share members.
- The entities in S are of the same type. For example, this definition does not allow people and organizations to be in the same community.
- By no means does this definition cover every aspect of communities in the real world. For example, it does not consider the temporal dimension of communities. Usually a community exists within a specific period of time. Similarly, an entity may belong to a community during some time periods.
- This is a conceptual definition. In practice, different community mining algorithms have their own operational definitions which usually depend on how communities manifest themselves in the given data (which we will discuss shortly). Furthermore, the algorithms may not be able to discover all the members of a community or its precise theme.

Communities may also have hierarchical structures.

Definition (sub-community, super-community, and sub-theme): A community (T, G) may have a set of **sub-communities** $\{(T_1, G_1), \ldots, (T_m, G_m)\}$, where T_i is a **sub-theme** of T and $G_i \subseteq G$. (T, G) is also called a **super-community** of (T_i, G_i). In the same way, each sub-community (T_i, G_i) can be further decomposed, which gives us a **community hierarchy**.

Community Manifestation in Data: Given a data set, which can be a set of Web pages, a collection of emails, or a set of text documents, we want to find communities of entities in the data. However, the data itself usually does not explicitly give us the themes or the entities (community members) associated with the themes. The system needs to discover the hidden community structures. Thus, the first issue that we need to know is how communities manifest themselves. From such manifested evidences, the system can discover possible communities. Different types of data may have different forms of manifestation. We give three examples.

Web Pages:

1. Hyperlinks: A group of content creators sharing a common interest is usually inter-connected through hyperlinks. That is, members in a community are more likely to be connected among themselves than outside the community.
2. Content words: Web pages of a community usually contain words that are related to the community theme.

Emails:

1. Email exchange between entities: Members of a community are more likely to communicate with one another.
2. Content words: Email contents of a community also contain words related to the theme of the community.

Text documents:

1. Co-occurrence of entities: Members of a community are more likely to appear together in the same sentence and/or the same document.
2. Content words: Words in sentences indicate the community theme.

Clearly, the key form of manifestation of a community is that its members are linked in some way. The associated text often contains words that are indicative of the community theme.

Objective of Community Discovery: Given a data set containing entities, we want to discover hidden communities of the entities. For each community, we want to find the theme and its members. The theme is usually represented with a set of keywords.

7.5.2 Bipartite Core Communities

HITS finds dense bipartite graph communities based on broad topic queries. The question is whether it is possible to find all such communities efficiently from the crawl of the whole Web without using eigenvector computation which is relatively inefficient. Kumar et al. [293] presented a technique for finding bipartite cores, which are defined as follows.

Recall that the node set of a bipartite graph can be partitioned into two subsets, which we denote as set F and set C. A **bipartite core** is a complete bipartite sub-graph with at least i nodes in F and at least j nodes in C. A complete bipartite graph on node sets F and C contains all possible edges between the vertices of F and the vertices of C. Note that edges within F or within C are allowed here to suit the Web context, which deviate from the traditional definition of a complete bipartite graph. Intuitively, the core is a small (i, j)-sized complete bipartite sub-graph of the community, which contains some core members of the community but not all.

The cores that we seek are *directed*, i.e., there is a set of i pages all of which link to a set of j pages, while no assumption is made of links out of the latter set of j pages. Intuitively, the former is the set of pages created by members of the community, pointing to what they believe are the most valuable pages for that community. For this reason we will refer to the i pages that contain the links as **fans**, and the j pages that are referenced as **centers** (as in community centers). Fans are like specialized *hubs*, and centers are like *authorities*. Figure 7.11 shows an example of a bipartite core.

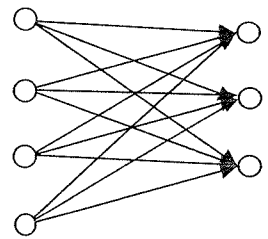

Fig. 7.11. A (4, 3) bipartite core

In Fig. 7.11, each fan page links to every center page. Since there are four fans and three centers, this is called a (4, 3) bipartite core. Such a core almost certainly represents a Web community, but a community may have multiple bipartite cores.

Given a large number of pages crawled from the Web, which is represented as a graph, the procedure for finding bipartite cores consists of two major steps: pruning and core generation.

Step 1: Pruning

We describe two types of pruning to remove those unqualified pages to be fans or centers. There are also other pruning methods given in [293].

1. **Pruning by in-degree:** we can delete all pages that are very highly referenced (linked) on the Web, such as homepages of Web portals (e.g., Yahoo!, AOL, etc). These pages are referenced for a variety of reasons, having little to do with any single emerging community, and they can be safely deleted. That is, we delete pages with the number of in-links great than k, which is determined empirically ($k = 50$ in [293]).
2. **Iterative pruning of fans and centers:** If we are interested in finding (i, j) cores, clearly any potential fan with an out-degree smaller than j can be pruned and the associated edges deleted from the graph. Similarly, any potential center with an in-degree smaller than i can be pruned and the corresponding edges deleted from the graph. This process can be done iteratively: when a fan gets pruned, some of the centers that it points to may have their in-degrees fall below the threshold i and qualify for pruning as a result. Similarly, when a center gets pruned, a fan that points to it could have its out-degree fall below its threshold of j and qualify for pruning.

Step 2: Generating all (i, j) Cores

After pruning, the remaining pages are used to discover cores. The method works as follows: Fixing j, we start with all $(1, j)$ cores. This is simply the set of all vertices with out-degree at least j. We then construct all $(2, j)$ cores by checking every fan which also points to any center in a $(1, j)$ core. All $(3, j)$ cores can be found in the same fashion by checking every fan which points to any center in a $(2, j)$ core, and so on. The idea is similar to the Apriori algorithm for association rule mining (see Chap. 2) as every proper subset of the fans in any (i, j) core forms a core of smaller size.

Based on the algorithm, Kumar et al. found a large number of topic coherent cores from a crawl of the Web [293]. We note that this algorithm only finds the core pages of the communities, not all members (pages). It also does not find the themes of the communities or their hierarchical organizations.

7.5.3 Maximum Flow Communities

Bipartite cores are usually very small and do not represent full communities. In this section, we define and find maximum flow communities based on the work of Flake et al. [180]. The algorithm requires the user to give a

set of seed pages, which are examples of the community that the user wishes to find.

Given a Web link graph $G = (V, E)$, a maximum flow community is defined as a collection $C \subset V$ of Web pages such that each member page $u \in C$ has more hyperlinks (in either direction) within the community C than outside of the community V-C. Identifying such a community is intractable in the general case because it can be mapped into a family of NP-complete graph partition problems. Thus, we need to approximate and recast it into a framework with less stringent conditions based on the network flow model from operations research, specifically the maximum flow model.

The maximum flow model can be stated as follows: We are given a graph $G = (V, E)$, where each edge (u, v) is thought of as having a positive capacity $c(u, v)$ that limits the quantity of a product that may be shipped through the edge. In such a situation, it is often desirable to have the maximum amount of flow from a starting point s (called the **source**) and a terminal point t (called the **sink**). Intuitively, the maximum flow of the graph is determined by the bottleneck edges. For example, given the graph in Fig. 7.12 with the source s and the sink t, if every edge has the unit capacity, the bottleneck edges are W-X and Y-Z.

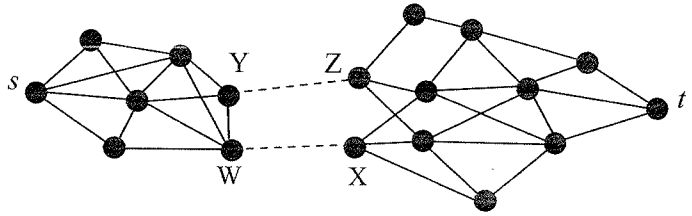

Fig. 7.12. A simple flow network.

The **Max Flow-Min Cut** theorem of Ford and Fulkerson [181] proves that the maximum flow of a network is identical to the minimum cut that separates s and t. Many polynomial time algorithms exist for solving the s-t maximum flow problem. If Fig. 7.12 is a Web link graph, it is natural to cut the edges W-X and Y-Z to produce two Web communities.

The basic idea of the approach in [180] is as follows: It starts with a set S of *seed* pages, which are example pages of the community that the user wishes to find. The system then crawls the Web to find more pages using the seed pages. A maximum flow algorithm is then applied to separate the community C involving the seed pages and the other pages. These steps may need to be repeated in order to find the desired community. Figure 7.13 gives the algorithm.

Algorithm Find-Community (S)
  while number of iteration is less than desired **do**
    build $G = (V, E)$ by doing a fixed depth crawl starting from S;
    $k = |S|$;
    C = Max-Flow-Community(G, S, k);
    rank all $v \in C$ by the number of edges in C;
    add the highest ranked non-seed vertices to S
  end-while
  return all $v \in V$ still connected to the source s

Procedure Max-Flow-Community(G, S, k)
  create artificial vertices, s and t and add to V; // V is the vertex set of G.
  for all $v \in S$ **do**
    add (s, v) to E with $c(s, v) = \infty$ // E is the edge set of G.
  endfor
  for all $(u, v) \in E$, $u \neq s$ **do**
    $c(u, v) = k$;
    if $(v, u) \notin E$ **then**
      add (v, u) to E with $c(v, u) = k$
    endif
  endfor
  for all $v \in V$, $v \notin S \cup \{s, t\}$ **do**
    add (v, t) to E with $c(v, t) = 1$
  endfor
  Max-Flow(G, s, t);
  return all $v \in V$ still connected to s.

Fig. 7.13. The algorithm for mining maximum flow communities

The algorithm Find-Community is the control program. It takes a set S of seed Web pages as input, and crawls to a fixed depth including in-links as well as out-links (with in-links found by querying a search engine). It then applies the procedure Max-Flow-Community to the induced graph G from the crawl. After a community C is found, it ranks the pages in the community by the number of edges that each has inside of the community. Some highest ranked non-seed pages are added to the seed set. This is to create a big seed set for the next iteration in order to crawl more pages. The algorithm then iterates the procedure. Note that the first iteration may only identify a very small community. However, when new seeds are added, increasingly larger communities are identified. Heuristics are used to decide when to stop.

The procedure Max-Flow-Community finds the actual community from G. Since a Web graph has no source and sink, it first augments the web

graph by adding an artificial source, s, with infinite capacity edges routed to all seed vertices in S; making each pre-existing edge bidirectional and assigning each edge a constant capacity k. It then adds an artificial sink t and routes all vertices except the source, the sink, and the seed vertices to t with unit capacity. After augmenting the web graph, a residual flow graph is produced by a maximum flow procedure (Max-Flow()). All vertices accessible from s through non-zero positive edges form the desired result. The value k is heuristically chosen to be the size of the set S to ensure that after the artificial source and sink are added to the original graph, the same cuts will be produced as the original graph (see the proof in [179]). Figure 7.14 shows the community finding process.

Finally, we note that this algorithm does not find the theme of the community or the community hierarchy (i.e., sub-communities and so on).

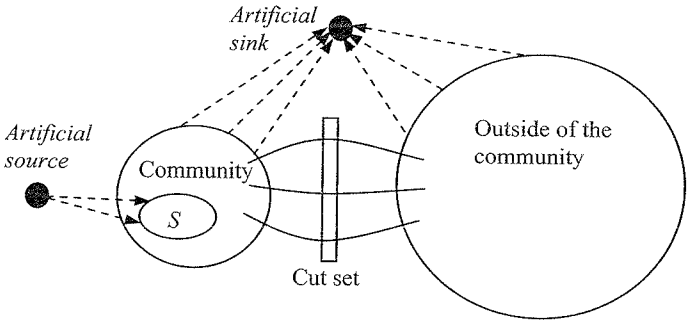

Fig. 7.14. Schematic representation of the community finding process

7.5.4 Email Communities Based on Betweenness

Email has become the predominant means of communication in the information age. It has been established as an indicator of collaboration and knowledge (or information) exchange. Email exchanges provide plenty of data on personal communication for the discovery of shared interests and relationships between people, which were hard to discover previously.

It is fairly straightforward to construct a graph based on email data. People are the vertices and the edges are added between people who corresponded through email. Usually, the edge between two people is added if a minimum number of messages passed between them. The minimum number is controlled by a threshold, which can be tuned.

To analyze an email graph or network, one can make use of all the centrality measures and prestige measures discussed in Sect. 7.1. We now focus on community finding only.

We are interested in people communities, which are subsets of vertices that are related. One way to identify communities is by partitioning the graph into discrete clusters such that there are few edges lying between the clusters. This definition is similar to that of the maximum flow community. **Betweenness** in social networks is a natural measure for identifying those edges in between clusters or communities [523]. The idea is that inter-community links, which are few, have high betweenness values, while the intra-community edges have low betweenness values. However, the betweenness discussed in Sect. 7.1 is evaluated on each person in the network. Here, we need to evaluate the betweenness of each edge. The idea is basically the same and Equation (4) can be used here without normalization because we only find communities in a single graph. The betweenness of an edge is simply the number of shortest paths that pass it.

If the graph is not connected, we identify communities from each connected component. Given a connected graph, the method works iteratively in two steps (Fig. 7.15):

repeat
  Compute the betweenness of each edge in the remaining graph;
  Remove the edge with the highest betweenness
until the graph is suitably partitioned.

Fig. 7.15. Community finding using the betweenness measure.

Since the removal of an edge can strongly affect the betweenness of many other edges, we need to repeatedly re-compute the betweenness of all edges. The idea of the method is very similar to the minimum-cut method discussed in Sect. 7.5.3.

The stopping criteria can be designed according to applications. In general, we consider that the smallest community is a triangle. The algorithm should stop producing more unconnected components if there is no way to generate triangle communities. A component of five or fewer vertices cannot consist of two viable communities. The smallest such component is six, which has two triangles connected by one edge, see Fig. 7.16. If any discovered community does not have a triangle, it may not be considered as a community. Clearly, other stopping criteria can be used.

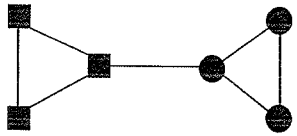

Fig. 7.16. The smallest possible graph of two viable communities.

7.5.5 Overlapping Communities of Named Entities

Most community discovery algorithms are based on graph partitioning, which means that an entity can belong to only a single community. However, in real life, a person can be in multiple communities (see the definition in Sect. 7.5.1). For example, he/she can be in the community of his/her family, the community of his/her colleagues and the community of his/her friends. A heuristic technique is presented in [325] for finding overlapping communities of entities in text documents.

In the Web or email context, there are explicit links connecting entities and forming communities. In free text documents, no explicit links exist. Then the question is: what constitutes a link between two entities in text documents? As we indicated earlier, one simple technique is to regard two entities as being linked if they co-occur in the same sentence. This method is reasonable because if two people are mentioned in a sentence there is usually a relationship between them.

The objective is to find entity communities from a text corpus, which could be a set of given documents or the returned pages from a search engine using a given entity as the search query. An entity here refers to the name of a person or an organization.

The algorithm in [325] consists of four steps:

1. Building a link graph: The algorithm first parses each document. For each sentence, it identifies named entities contained in the sentence. If a sentence has more than one named entities, these entities are pair-wise linked. The keywords in the sentence are attached to the linked pairs to form their textual contents. All the other sentences are discarded.
2. Finding all triangles: The algorithm then finds all triangles, which are the basic building blocks of communities. A triangle consists of three entities bound together. The reason for using triangles is that it has been observed by researchers that a community expands predominantly by triangles sharing a common edge.
3. Finding community cores: It next finds community cores. A community core is a group of tightly bound triangles, which are relaxed complete sub-graphs (or cliques). Intuitively, a core consists of a set of tightly connected members of a community.
4. Clustering around community cores: For those triangles and also entity pairs that are not in any core, they are assigned to cores according to their textual content similarities with the discovered cores.

It is clear that in this algorithm a single entity can appear in multiple communities because an entity can appear in multiple triangles. To finish off, the algorithm also ranks the entities in each community according to degree centrality. Keywords associated with the edges of each community are also ranked. The top keywords are assumed to represent the theme of the community. The technique has been applied to find communities of political figures and celebrities from Web documents with promising results.

Bibliographic Notes

Social network analysis has a relative long history. A large number of interesting problems and algorithms were studied in the past 60 years. The book by Wasserman and Faust [540] is an authoritative text of the field. Co-citation [494] and bibliographic coupling [275] are from bibliometrics, which is a type of research method used in library and information science. The book edited by Borgman [58] is a good source of information on both the research and applications of bibliometrics.

The use of social network analysis in the Web context (also called link analysis) started with the PageRank algorithm proposed by Brin and Page [68] and Page et al. [422], and the HITS algorithm proposed by Kleinberg [281]. PageRank is also the algorithm that powers the Google search engine. Due to several weaknesses of HITS, many researchers have tried to improve it. Various enhancements were reported by Lempel and Moran [310], Bharat and Henzinger [52], Chakrabarti et al. [88], Cai et al. [78], etc. The book by Langville and Meyer [304] contains in-depth analyses of PageRank, HITS and many enhancements to HITS. Other works related to Web link analysis include those in [98, 226, 266, 368] on improving the PageRank computation, in [168] on searching workspace Web, in [103, 182, 183, 416] on the evolution of the Web and the search engine influence on the Web, in [140, 142, 410, 516] on other link based models, in [34, 440, 370, 371] on Web graph and its characteristics, in [37, 51, 235] on sampling of Web pages, and in [32, 425, 585] on the temporal dimension of Web search.

On community discovery, HITS can find some communities by computing non-principal eigenvectors [198, 281]. Kumar et al. [293] proposed the algorithm for finding bipartite cores. Flake et al. [179] introduced the maximum flow community mining. Ino et al. [249] presented a more strict definition of communities. Tyler et al. [523] gave the method for finding email communities based on betweenness. The algorithm for finding overlapping communities of named entities from texts was given by Li et al. [325]. More recent developments on communities and social networks on the Web can be found in [16, 21, 137, 158, 200, 518, 519, 561, 618].