Chapter 2

Search Engine Technology

In the last several years, the World Wide Web has become the most popular place for people to publish and obtain information. It is estimated that there are currently over three billion publicly accessible web pages on the Web. As more and more web pages are added to the Web, how to find useful information quickly has quickly become a challenge for millions of Web users. Currently, there are primarily two approaches to find desired data on the Web.

Browsing A user can start the browse from a starting page and follow certain hyperlinks to navigate to other pages. The activity of following hyperlinks can be repeated until the user has either found the desired information or become bored. A key to successful browsing is to find a good starting page. To facilitate browsing, web pages can be organized into a category hierarchy such as the one provided by Yahoo (www.yahoo.com). In a typical category hierarchy, the root contains very general categories such as *education* and *sports*. Each category in turn contains less general categories. This hierarchy helps narrow down the information space gradually and eventually leads to desired information. A portion of an example category hierarchy for sports is shown in Figure 2.1.

Searching A user can utilize a search engine to find desired information. From the interface of a search engine, the user submits a query that describes the user's information needs. When the query is processed by the search engine, a list of documents are returned to the user, usually

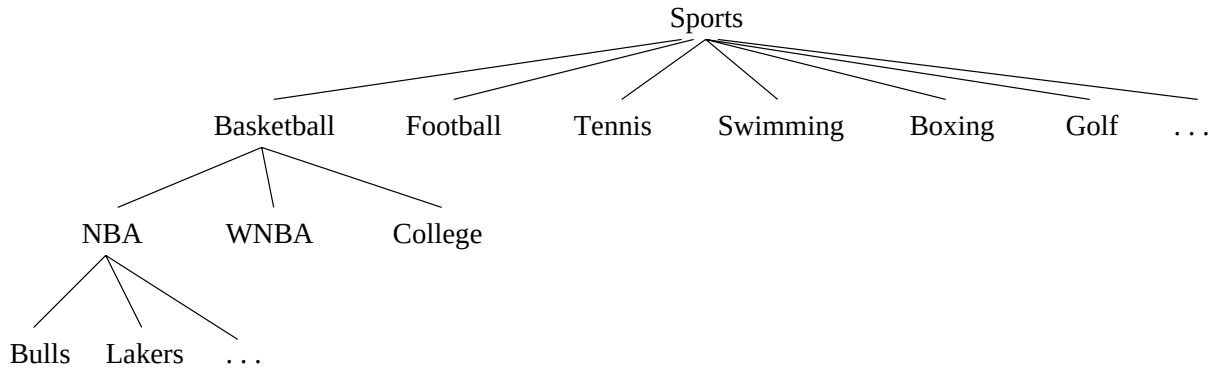

Figure 2.1: An Example Category Hierarchy for Sports

with the best matched documents displayed at the top. Many search engines are running on the Web. Some of the most well known ones are AltaVista, WebCrawler, HotBot, Google and Lycos. To enable fast evaluation of user queries, most search engines preprocess documents searchable in their engines by creating an inverted file index for these documents.

Each of the above two approaches has advantages and disadvantages when compared with the other. Browsing tends to yield higher quality result (i.e., fewer junk pages) when a good starting page is used. The category hierarchy often used to guide browsing may need to be created mostly manually, requiring a lot of manpower. As a result, only a very small portion of web pages can be categorized. In addition, navigating through the hierarchy can be tedious and a user can sometimes get lost during the process due to several factors: (1) The hierarchy usually becomes very complex after the first several levels and the user may not know which subhierarchy to navigate next. (2) The desired data may be placed under different subhierarchies. (3) The user may get confused by the hierarchy if the way the data are organized does not match what is expected by the user; this is possible because different people may have different ways to categorize data (i.e., categorization is highly subjective). In contrast, a search engine can automatically index a large number of web pages and as a result, a single query can often yield a large number of returned pages. A main problem with current search engines is that users often get overwhelmed by a high percentage of junk pages in the search result. Nevertheless, using a search engine is a quick way to obtain desired information on the Web and millions of users use search engines everyday.

A basic search engine is essentially a text retrieval system on the Web for searching web pages. The basic technologies used to build text retrieval systems as discussed in Chapter 1 are also used to build search engines. Nevertheless, there are significant differences between the environment where traditional text retrieval systems are developed and the Web environment where search engines are built. These differences have prompted new and exciting technologies be developed and incorporated into advanced search engines. In this chapter, we describe these new technologies. But before we do that, we would like to first point out the unique characteristics of the Web environment over traditional text retrieval environment and lay out the reason why these characteristics motivate us to develop new technologies for search engines.

Unique characteristics of the Web environment:

1. Web pages are widely distributed on many servers. Traditional text retrieval systems usually assume that all documents in a system are provided at a central site and they are under the control of the system. Any change (addition, deletion and modification) to a document must go through the system. In contrast, in the Web environment, pages are stored at numerous autonomous Web servers. In order to index these pages, we first need to fetch them to the site of the search engine. This is usually accomplished by a program known as *Web robot* or *Web spider*. In addition, the search engine has no control on changes to Web pages. As a result, in order to keep the index data of the search engine reasonably up-to-date, the robot program needs to visit different Web servers periodically to refetch changed pages and/or fetch newly added pages. In Section 2.1, we describe issues related to the creation of the robot program.

2. Web pages are highly tagged documents. At present, most Web pages are in HTML (HyperText Markup Language) format. In the foreseeable future, XML (eXtensible Markup Language) documents may be widely used. These tags often convey rich information regarding the terms used in documents. For example, a term appearing in the title of a document

or a term emphasized with special font can provide hint that the term is rather important in indicating the contents of the document. Traditional text retrieval systems are usually based on plain text documents that are either not or rarely tagged. In Section 2.2, we discuss some techniques that utilize these tags to improve retrieval effectiveness.

3. In the Web environment, pages are extensively linked. A link from page A to page B provides a convenient path for a Web user to navigate from page A to page B. Careful analysis can reveal that such a simple link could contain several pieces of information that may be made use of to improve retrieval effectiveness. First, such a link indicates a good likelihood that the contents of the two pages are related. Second, the author of page A values the contents of page B. Note that page B could be one of a number of possible pages that have similar contents. The fact that page B is chosen indicates that the author of page A sees something special about page B. Third, when the author of page A creates the link to page B, he or she often attaches a few words to the link to indicate the contents of page B. The set of words associated with the link is called an *anchor* of page B and each word in the set is called an *anchor term*. Documents in traditional text retrieval systems are usually not linked. In Section 2.3, we introduce several methods for utilizing linkage Information among Web pages to improve the search engine retrieval performance.

4. A large search engine could be used by hundreds of thousands of users each day. These users may submit even a larger number of queries every day. Many users may share similar search interests and/or even submit identical or similar queries. In addition, the same user may use the same search engine multiple times over a period of time. Rich knowledge can be discovered from different users' search histories and reactions (*filtering behaviors* such as clicking and viewing) to returned documents. In Section 2.4, we discuss how to derive useful knowledge from users' interaction with a search engine and use the knowledge to help the users of the search engine obtain better results.

2.1 Web Robot

A Web robot (also known as *spider, crawler* and *wanderer*) is a program for fetching Web pages from the Web. Robots can be used in several types of applications such as Web site analysis and the creation of mirror sites. In this book, robots are used to fetch Web pages to the site of a search engine. After all Web pages needed by a search engine are fetched, they are preprocessed to create an index database for the search engine to use when evaluating user queries.

The main idea behind the robot program is quite simple. Note that each page on the Web has a URL (Universal Resource Locator) which identifies the location of the page on the Web. First, one or more initial URLs are placed into a URL queue. Next, the following steps are repeated until either no new URLs can be found or enough pages have been fetched: (1) take the next URL from the queue and fetch the corresponding Web page from its server to a local site using the HTTP (HyperText Transfer Protocol); (2) from the fetched Web page, extract new URLs and add them to the queue.

When actually implementing a robot, the following issues need to be addressed.

1. Which initial URL(s) to use?

The answer depends on what document collection we would like to generate for a search engine. If the goal is to build a general-purpose search engine such as AltaVista and Lycos, then URLs that are likely to reach a large portion of the Web should be used. One such URL is the URL of the Yahoo homepage. If the goal is to build a search engine for a local organization, then the homepage of the organization would be a good choice. As an example, suppose we would like to collect all Web pages related to Binghamton University. In this case, the URL of the homepage of the university, www.binghamton.edu, can be used. It is likely that some Web pages from the Binghamton University domain have links to pages outside the domain. As a result, Web pages that are not related to Binghamton University may be fetched by the robot. This problem can be solved easily by checking whether a new URL is in the correct domain (e.g., check if it contains "binghamton.edu") and add a new URL to the queue only if it satisfies the domain requirement.

2. How to extract correct URLs from a Web page?

   We need to identify all possible HTML tags and tag attributes that can hold URLs. While most URLs appear in the anchor tag (e.g., ...), there are other tags in which URLs could appear. For example, a URL may appear in the option tag as in <option value="URL" ...> ... </option>, in the area tag (map) as in <area href="URL" ...> ... </area>, or in the frame tag as in <frame src="URL" ...> ... </frame>. Frequently a URL appearing in a Web page P does not contain the full path needed to locate the corresponding Web page from a browser. Instead, a partial path (relative path) is used and a full path can be derived from either the location of P or a base path provided in P.

3. How fast should a robot fetch Web pages from a server?

   Common sense seems to suggest that a robot should fetch Web pages as fast as possible. The problem is actually more complex than this. When a robot fetches a Web page from a remote server, it will consume some of the server's CPU/IO resources. When a large number of Web pages from the same server are fetched in high speed, the remote server could be overwhelmed. A good robot designer should be considerate to remote servers from which Web pages are fetched. Ideally, the operation of a robot should not affect remote servers in a noticeable way. This can be achieved by slowing down the speed of fetching from the same server. For example, the robot can wait for a few minutes before fetching another Web page from the same server. Since a robot usually fetches Web pages from a large number of Web servers, it can alternate the fetching from different servers so that the robot can be kept busy while all remote servers feel little impact.

Most robots fetch Web pages in a breadth-first order from the initial URLs. This order works quite adequately when the objective of the robot is to fetch as many Web pages in specified domain(s) as possible. There are situations where fetching Web pages in a different order may be preferred. For example, we may have limited resources such as limited storage space or limited time so we want to fetch as many *important* pages as possible subject to the limited resources. Here the *importance* of a page can be defined in a number of ways such as its popularity (e.g., the number of pages that have links pointing to it) or whether it is related to an area of interest. Fetching Web pages that are related to a specified subject area is very important in building domain-specific search engines. Here the term "domain" refers to application domain such as movie or sport rather URL domain.

One way to fetch as many pages related to a given subject as possible quickly is to identify promising URLs (i.e., URLs that are likely to lead to related pages) in the URL queue and use these URLs first. Often heuristics can be used to identify promising URLs. As an example, suppose we are interested in obtaining Web pages that are related to "movie". More specifically, suppose a query consisting of one or more related terms such as "movie" and "motion picture" is used and a page is considered to be relevant (or *hot*) to the query if its similarity with the query is above a given threshold. The following heuristics can be used to identify potential hot pages. A page has a high likelihood to be hot if one of the following is true:

1. An anchor of its URL contains one or more query terms.

2. Its URL contains one or more query terms.

3. Its URL is within 3 links away from a hot page. Several studies have observed that related pages tend to be clustered together. This heuristic reflects this observation.

Based on the above heuristics, the robot program can be revised to consider two types of URLs, namely *hot URLs* — URLs that satisfy the heuristics and *non-hot URLs*. Accordingly, two URL queues are used, one is hot_queue for hot URLs whose pages have not been fetched and the other is url_queue for non-hot URLs whose pages have not been fetched. When the robot needs to fetch another Web page, it first considers URLs from the hot_queue. The url_queue is considered only when the hot-queue is empty. The modified robot can obtain hot pages more quickly than a regular robot. The following procedure summarizes the above discussion [5].

Priority-Crawling

```
begin
    hot_queue = url_queue = empty; /* initialization */
    enqueue(url_queue, starting_url)
    while (hot_queue or url_queue is not empty)
          { url = dequeue(hot_queue, url_queue) /* dequeue hot_queue first if it is not empty */
            page = fetch(url);
            if (page is hot)
                  hot[url] = true; /* the url is a hot url */
            enqueue(crawled_urls, url); /* crawled_urls contains urls of pages that have been fetched
*/
            url_list = extract_urls(page); /* extracting URLs from the fetched page */
            for each u in url_list
                  if (u not in url_queue and u not in hot_queue and u not in crawled_urls)
                        /* u is a new url */
                      if (u is a hot URL) /* the three heuristics are examined */
                            enqueue(hot_queue, u);
                      else enqueue(url_queue, u);
          };
end;
```

2.2 Use of Tag Information

Web pages are usually formatted in certain markup language, mostly noticeably in HTML. A markup language formats a document through the use of a number of tags such as *title* and *list*. Usually, tags appear in pairs with one indicating the start and the other indicating the end. For example, in HTML, the starting title tag is <title> and the ending title tag is </title>. Currently, tag information is used in primarily two different ways to improve retrieval effectiveness. First, tag information can be used to select index terms, i.e., choosing terms to represent a document. Second, tag information can be used to determine the degrees of significance or weights of index terms.

In traditional text retrieval systems, other than stop words, all terms in a document are used to index the document. In Web search engines, we may choose not to use certain terms in a document to index the document. For example, in order to make its index database more scalable against the fast growing Web, Lycos uses only "important" terms as index terms. An "important" term could be a term in the title, in one of the headers, in a location near the start or the end of the page, or a term that has a high frequency or is emphasized with an emphatic font such as boldface and large sized font. In contrast, terms that appear in the middle of the page and in reduced fonts such as small and tiny fonts can be considered not or less important. The practice of using only selected terms to index documents is called *partial-text indexing* and the regular approach of using all terms is called *full-text indexing*. It is clear that tag information is very useful for identifying "important" terms.

On the other hand, we can use certain terms not in a document to represent the document according to our needs. As we mentioned previously, when a page A has a link to page B, a set of terms known as *anchor terms* is usually associated with the link. The purpose of using the anchor terms is to provide information regarding the contents of page B to facilitate navigation by human users. The anchor terms often provide related terms or synonyms to the terms used to index page B. To utilize such valuable information, several search engines like Google [1] and WWWW [15] have suggested to also use anchor terms to index linked pages (e.g., page B). In general, a Web page may be linked by many other Web pages and has many associated anchor terms. Anchor terms can also be used as collateral text to index non-textual objects such as images.

In Chapter 1, we mentioned that the significance (weight) of a term in a document may depend on its term frequency (the number of times that the term appears in the document) and its document frequency (the number of documents having the term). Tag information can also be used to influence the weight of a term. For example, many well-known search engines such as AltaVista, HotBot and Yahoo have been known to assign higher weights to terms in the title. Terms in large fonts or special fonts such as boldface, italics and underscored may also be assigned higher weights.

A more general approach for determining the relative weights of term occurrences in different tags is as follows [7]. First, the set of available tags is partitioned into a number of subsets. For example, the title tag could be a subset by itself, all header tags (HTML has six levels of headers h1, ..., h6) could form a subset, all list tags (HTML has three different list tags, namely ul for unordered list, ol for ordered list and dl for descriptive list) can form a separate subset, all emphatic tags (large fonts and special fonts) can be grouped together as a subset and the rest of the tags can form yet another subset. Next, based on the partition of the tags, term occurrences in a document can be partitioned into a number of classes. For example, all term occurrences appearing in headers form a class. In addition to these classes, two special classes can be formed. The first contains

terms in plain text (i.e., with no tags) and the second contains anchor terms associated with the links pointing to the page. Let n be the number of classes formed. With these classes, the term frequency of each term can be represented as a *term frequency vector*: $tfv = (tf_1, ..., tf_n)$, where tf_i is the number of times the term appears in the ith class, $i = 1, ..., n$. Finally, we can assign different importance to different classes. Let $civ = (civ_1, ..., civ_n)$ be the *class importance vector* such that civ_i is the importance assigned to the ith class, $i = 1, ..., n$. With tfv and civ, the traditional term frequency weight formula can be extended into $tf_1 * civ_1 + \cdots + tf_n * civ_n$. This formula takes both the frequencies of the term in different classes as well as the importance of different classes into consideration. Note that when the civ for the anchor class is assigned 0 and all other civ's are assigned 1, $tf_1 * civ_1 + \cdots + tf_n * civ_n = tf$ is the total frequency of the term in a document.

An interesting question is which class importance vector can provide the best performance improvement over the traditional method in which term occurrences are not partitioned into classes. Let the civ that can achieve the most improvement be called an *optimal civ*. One method to find an optimal or near optimal civ is as follows. First, a test bed is utilized. The test bed contains a Web page collection and a set of queries. In addition, for each query, the set of relevant documents has been identified. Next, different civ's can be tested and the civ that yields the highest retrieval effectiveness for the testbed queries can be considered as an optimal civ. As there may be infinite number of different civ's, heuristic algorithms such as genetic algorithm may be employed to find an optimal or near optimal civ efficiently and automatically [8].

2.3 Use of Linkage Information

As we mentioned earlier, Web pages are extensively linked. The linkage information can be utilized in a number of ways to improve retrieval performance. In the previous section, we already saw that anchor terms associated with links could be used to index linked pages. Other methods of utilizing linkage information are discussed below.

2.3.1 Vector Spread Activation

A page, say page A, is said to be a parent of another page, say page B, if page A has a link to page B. Often, for a given query, if a page p has many relevant parent pages, then p is also likely to be relevant even though page p itself may have low similarity with the query. Vector Spread Activation [21] is a method that is designed to increase the chance that the page p is retrieved by a query q by adding a portion of the similarities of its parents with q to its similarity with q. Suppose page p indeed has a low similarity with a query q. Clearly, if documents are retrieved based on their similarities directly, then p is unlikely to be retrieved due to its low similarity. On the other hand, relevant pages are likely to have high similarities on the average. Consider pages p_1 and p_2 such that p_1 is linked to by many relevant pages but p_2 is not. With vector spread activation, the new similarity of p_1 with respect to q is likely to be higher than that of p_2 due to the larger increase of similarity by the parents of p_1. As a result, if documents are retrieved based on the new similarities, then p_1 would have a better chance to be retrieved than p_2.

More precisely, vector spread activation works as follows. For a given query q, let $sim(q, p_i)$ be the regular similarity between q and Web page p_i. Suppose $link(x, y)$ is a function such that $link(x, y) = 1$ if page p_x has a link to page p_y and $link(x, y) = 0$ if p_x does not have a link to p_y.

The vector spread activation method ranks Web pages in descending values of $rs(q, p_i)$, the ranking score of p_i with respect to q, being defined by the following formula:

$$rs(q, p_i) = sim(q, p_i) + \alpha \sum_j link(j, i) * sim(q, p_j) \tag{2.1}$$

where α is a constant parameter defining the portion of a parent's similarity that needs to be propagated to a child page. The parameter is usually small such as 0.1. Note that $rs(q, p_i)$ is the "new similarity" of p_i with q in the above example.

Example 2.1 Suppose the regular similarities of three pages p_1, p_2 and p_3 with the query q are 0.4, 0.2 and 0.2, respectively, and pages p_1 and p_2 point to page p_3. When $\alpha = 0.1$, $rs(q, p_3) = sim(q, p_3) + 0.1(sim(q, p_1) + sim(q, p_2)) = 0.2 + 0.1 * (0.4 + 0.2) = 0.26$. ■

2.3.2 PageRank

The Web can be viewed as a gigantic directed graph $G(V, E)$, where V is the set of pages (vertices) and E is the set of hyperlinks (directed edges). Each page may have a number of outgoing edges (forward links) and a number of incoming edges (backlinks). As mentioned earlier, when an author places a link in page A pointing to page B, the author shows his/her belief that page B is at least somewhat important (i.e., worth pointing to). The Web graph contains a huge number of links and each link implies certain degree of importance of the pointed page. A very interesting question is how to use the links in the Web graph to measure the relative importance of each Web page. *PageRank* or simply *rank* is proposed to measure the relative importance of each page on the Web [18]. The rank of each page is computed based on the following three observations.

1. The larger the number of backlinks a page has, the more important the page is likely to be. This is intuitively correct as more backlinks imply that more authors consider the page to be important. In other words, the rank of a page reflects the popularity of the page among all Web page authors. The Yahoo homepage is probably one of the most important pages on the Web and it is probably linked by millions of pages.

2. A page can be an important page if it is linked by important pages even though there aren't a large number of pages linking to it. Intuitively, important pages are likely to be published by important authors or organizations and their endorsement should have more weight in determining the importance of a page. For example, if a page is pointed to by Yahoo homepage, then the page is likely to be important. The rank of a page can be considered as a weighted popularity measure of the page.

3. The more the forward links a page has, the less the influence the page should have on the importance of a pointed page. Item 2 above indicates that the importance of a page may be propagated to its child pages. If a page has multiple child pages, then these pages should share the importance of the parent page. As a result, if a page has more child pages, then it can only propagate a smaller fraction of its importance to each child page.

More formally, the rank is defined as follows. For a given Web page u, let F_u denote the set of pages u points to and B_u denote the set of pages that point to u. The rank of u, denoted $R(u)$, is defined by the formula below:

$$R(u) = \sum_{v \in B_u} \frac{R(v)}{|F_v|} \tag{2.2}$$

where $|F_v|$ denotes the number of pages in set F_v. Notice how the formula incorporates the three observations discussed above. First, the sum reflects that more backlinks can lead to larger rank. Second, that $R(v)$ is in the numerator indicates that the rank of u is increased more if page v is more important (i.e., has large $R(v)$). Third, that $|F_v|$ is in the denominator implies that the importance of a page is evenly divided and propagated to each of its child pages. Also notice that Formula (2.2) is a recursive formula. The computation can be carried out as follows. First, all pages are assigned with the same initial rank, say $1/N$, where N is the number of Web pages. This initial assignment keeps the sum of all ranks to be 1. Next, the formula is applied to compute the rank in a number of iterations. In each iteration, the rank of each page is computed using the ranks of its parent pages in the previous iteration. This is repeated until the ranks of the pages converge within a given threshold. Let $R_i(u)$ denote the rank of page u after the i-th iteration and $R_0(u)$ denote the initial rank assigned to page u. Then Formula (2.2) can be re-written as:

$$R_i(u) = \sum_{v \in B_u} \frac{R_{i-1}(v)}{|F_v|} \tag{2.3}$$

Formula (2.3) can be expressed in matrix format as follows. Let M be an $N \times N$ matrix representing the Web graph. If page v has a link to page u, then let the matrix entry m_{uv} be $1/|F_v|$. If there is no link from page v to page u, then $m_{uv} = 0$. Let R_i be an $N \times 1$ vector representing the rank vector of the N pages after the i-th iteration. Then Formula (2.3) can be expressed as:

$$R_i = M \times R_{i-1} \tag{2.4}$$

where R_0 is the initial rank vector with all entries having value $1/N$. When the ranks converge, the rank vector is the eigenvector of the matrix M with the corresponding eigenvalue being 1. Note that if every page has at least one forward link, then the sum of the values in each column of M is 1 and all values are non-negative (such a matrix is called a *stochastic* matrix). Looking from a different angle, the entries of matrix M can be interpreted as follows. Consider a surfer who is surfing the Web. At each step, the surfer follows a random link from the current page to one of its child pages. Thus the value at entry m_{uv} can be interpreted as the probability that a random walk from page v to its child pages will lead to page u. Now consider the effect on the ranks of pages by applying Formula (2.4). Suppose page v has a number of children. When Formula (2.4) is applied, the rank of v is propagated to its children. If each page has at least one forward link, the ranks of all pages will be passed on to their children. Since the sum of the ranks of all pages is initially 1, the sum is preserved at each iteration. Suppose by repeatedly applying Formula (2.4), the ranks of the pages converge. The converged rank of a page can be interpreted as the probability that the page will be visited by a random walk on the Web graph.

There is a problem with using the Formula (2.4) directly. That is, the ranks are guaranteed to converge only if M is *aperiodic* (i.e., the Web graph is not a large cycle) and *irreducible* (i.e., the Web graph is not *strongly connected*) [10, 17]. While the former (i.e., aperiodicity) is practically guaranteed for the Web, the latter is usually not true. A directed graph is said to be strongly connected if for any two distinct vertices in the graph, there is a directed path from one vertex to the other and vice versa. When the Web graph is not strongly connected, there may be pages (or a set of pages involved in a cycle) that only have backlinks but no forward links. These pages, which can only receive rank propagation from their parents but cannot propagate ranks to other pages, are called *rank sink*. The existence of rank sink can cause the loss of total rank value. One way to solve this problem is to conceptually add a link from each page to every page in the Web graph and associate an appropriate positive probability with each such a link [10]. The probabilities should be assigned in such a way that the matrix corresponding to the resulted Web graph has the stochastic property (i.e., the sum of all entries in each column is 1). Suppose a link is conceptually added from page v to page u with probability p. In the random walk model, this can be interpreted as that the Web surfer, while at page v, may jump to page u with probability p. Note that v and u may be the same page. In this case, the jump can be interpreted as a *refresh* or *reload* request.

Let us now consider how to assign probabilities to added links. Consider the following two cases for a given page v.

1. *Page v has no forward links in the original Web graph.* In this case, the Web surfer can only follow one of the added links (i.e., can only jump). It is reasonable to assume that the Web surfer may jump to any page with the same probability. Therefore, each added link from page v should have the probability $1/N$, where N is the number of pages in the Web graph. This is equivalent to making all the entries in the column corresponding to page v in the matrix be $1/N$.

2. *Page v has at least one forward link in the original Web graph, i.e., $|F_v| \geq 1$.* Based on matrix M, each such link is assigned a probability of $1/|F_v|$. This probability needs to be modified because without change the probabilities for the newly added links can only be zero, indicating no jumps are possible. Let c be a weight parameter satisfying $0 < c < 1$. Then we may adjust the probability for each original link from $1/|F_v|$ to $c * 1/|F_v|$ while assign probability $(1 - c) * 1/N$ to each newly added link from v. The closer to 1 the value of c is, the smaller the impact of the added links will be. It is easy to see that the sum of these probabilities is 1.

Mathematically, the addition of the links, the assignment of the probabilities to newly added links and the adjustment of the probabilities on original links can be done by modifying the matrix M to the following new matrix:

$$M^* = c \times (M + Z) + (1 - c) \times K \tag{2.5}$$

where Z is an $N \times N$ matrix such that all entries in the column corresponding to page v are either $1/N$ if v has no forward links in the original graph or zero if v has at least one forward link; K is an $N \times N$ matrix with all entries having the value of $1/N$; and c is a constant between 0 and 1.

When matrix M in Formula (2.4) is replaced by the new matrix M^*, the problem associated with rank sinks will be solved. Efficient techniques for computing PageRanks are discussed in [10].

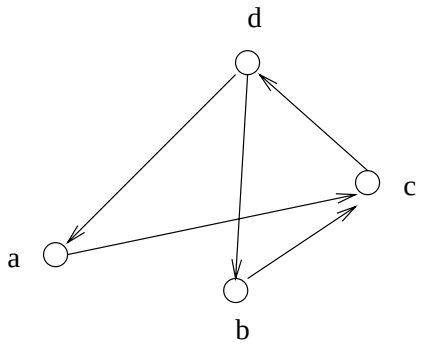

Figure 2.2: A Sample Web Graph

Finally, after convergence, the PageRanks of pages can be normalized by dividing the PageRank of each page by the maximum PageRank of all pages so that the normalized PageRank of any page is between 0 and 1. In this case, a page with PageRank = 1 is considered to be the most important document and a page with PageRank = 0 is considered to be least important.

Example 2.2 Consider the directed graph in Figure 2.2. Suppose nodes in the graph correspond to Web pages and directed edges denote links. We now compute the rank of each page.

Based on this graph, we have

$$M = \begin{pmatrix} 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Since each node has at least one forward link, all entries in matrix Z are zero. Since there are 4 vertices, all entries in matrix K are 1/4. Suppose the constant c in Formula (2.5) is 0.8. Then the new matrix M^* is

$$M^* = 0.8 \times (M + Z) + 0.2 \times K = \begin{pmatrix} 0.05 & 0.05 & 0.05 & 0.45 \\ 0.05 & 0.05 & 0.05 & 0.45 \\ 0.85 & 0.85 & 0.05 & 0.05 \\ 0.05 & 0.05 & 0.85 & 0.05 \end{pmatrix}$$

Suppose all pages have the same initial rank of 0.25, i.e., $R_0 = (0.25, 0.25, 0.25, 0.25)_t$, where V_t denotes the *transpose* of vector V. After 30 iterations, we have the following converged ranks $R(a) = R(b) = 0.176$, $R(c) = 0.332$ and $R(d) = 0.316$. Note that page c is pointed to by 2 pages, while each of the other pages is pointed at by 1 page only. As a result, the rank of c is higher than that of other pages. The reason why the rank of d is higher than that of page a or that of page b is that d is pointed at by the most important page, namely page c. ■

The PageRanks of Web pages can be used by a search engine to improve the retrieval effectiveness. The Google search engine (www.google.com), for example, uses PageRanks to improve the result of its search. We now describe a method for the PageRanks to be utilized by a search engine. For a given query, most search engines rank documents according to the similarities of

these documents with respect to the query. The similarity functions employed usually do not take linkage information into consideration. One way to utilize PageRanks is to define a new function for ranking documents that combines both PageRanks and similarities of documents.

Let $sim(q, d)$ be the content-based similarity between query q and document d, and $R(d)$ be the normalized PageRank of d. We now define a new quantity to measure the degree of relevance of document d with respect to q. The new quantity, denoted by $rel(q, d)$, is a weighted sum of $sim(q, d)$ and $R(d)$. More precisely,

$$rel(q,d) = \begin{cases} w * sim(q,d) + (1-w) * R(d), & if\ sim(q,d) > 0 \\ 0, & otherwise \end{cases} \tag{2.6}$$

where w is a constant between 0 and 1, indicating the importance of similarity relative to PageRank. With this function, when the similarity between a document and the query is 0, then the document will not be considered for retrieval as its degree of relevance is zero. In other words, the linkage information will be incorporated with the similarity function only when the similarity of the document is strictly positive. This is to avoid retrieving documents that have extremely high PageRanks but are unrelated to the query.

For each query, the search engine displays documents in descending order of their degrees of relevance. Now let us use an example to illustrate how this new ranking function can help retrieve better documents. Suppose that a user query asks for "IBM". There are numerous pages having the term "IBM", but the page most likely to be of interest to the user is the home page of IBM. Among the pages having the index term "IBM", the homepage of IBM is likely to be linked by a large number of other Web pages, i.e., its PageRank is likely to be very high among the pages that contain "IBM". As a result, the rank of the IBM homepage in the result is likely to be improved relative to other pages that have lower PageRanks.

2.3.3 Hub and Authority

PageRank defines the global importance of Web pages but the importance is domain/topic independent. Although the global importance is very useful, we often also need domain/topic dependent importance. For example, if you are a basketball fan, then you may be interested in important basketball pages. A topic dependent important page can be considered as an authoritative page with respect to a given topic. The question here is how to find authoritative pages for a given topic. Jon Kleinberg proposed to use authority scores to measure the importance of a Web page with respect to a given topic [12]. He also suggested a procedure to find topic specific important pages. The main ideas of this approach is reviewed in this subsection.

Web pages may be conceptually classified into two types: *Authoritative Pages* that contain informative contents on one or more topics, and *hub pages* that have links to authoritative pages. It is observed that for a given topic, a good authoritative page is often linked by many good hub pages related to the topic and a good hub page often has links to many good authoritative pages related to the topic. In other words, good authoritative pages and good hub pages related to the same topic often reinforce each other. To draw an analogy, suppose each page is a research paper. Then a good authoritative page is like an important paper on a topic and a good hub page is like a good survey paper/article on the same topic. Note that it is possible for the same page to be both a good authoritative page and a good hub page on the same topic.

In the Web search engine environment, a topic can be defined by a user query. The procedure proposed in [12] for finding good authoritative and hub pages can be summarized into the following three steps. Let q be a user submitted query.

1. The query is first processed by a typical similarity-based search engine. After this step, a set S of documents with the largest similarities is returned. The set S is called the *root set*. In order to find good authoritative pages with respect to a query, the size of S should be reasonably large, say in the low hundreds.

2. Expand the root set S into a larger set T called the *base set*. T is expanded from S as follows. Initially, $T = S$. Next, any page that is pointed to by a page in S is added to T. Second, any page that has a link to a page in S is added to T. In other words, T contains all pages in S as well as all the parent pages and child pages of the pages in S. In order to compute T from S quickly, the search engine can save the link structure of the Web in advance. The link structure can be stored in a table with two columns (parent_url, child_url) which can be constructed by the Web robot when collecting Web pages for the search engine.

   Note that while a Web page typically has limited number of child pages, it may have an extremely large number of parent pages. For example, the homepage of Yahoo may be linked by millions of pages. In order to limit the size of the base set, a threshold say 20 can be used such that at most 20 parent pages of each page in S are included in T. Similar restriction may also be applied to the inclusion of child pages. The 20 parent pages could be chosen randomly from all the parent pages of a page in S. Some heuristics may be applied to choose better parent pages. As mentioned previously, when a page has a link to another page, anchor text is associated with the link. One heuristic is to select those parent pages whose corresponding anchor text contains many terms in the query.

   The base set typically contains thousands of pages. If S contains 200 pages and we include 20 parent pages and 20 child pages for each page in S, then the base set could have up to 8,200 pages (some duplicate pages may exist). It is expected that the base set for a given query contains sufficient number of pages related to the query. In the subsequent steps, our task is to identify the most authoritative pages with respect to the query from the base set.

3. Compute the authority score and hub score of each page in the base set T. For a given page p, let $a(p)$ and $h(p)$ be the authority and hub scores of p, respectively. Initially, $a(p) = h(p) = 1$ for each page p. For pages p and q, let (p, q) denote a link from p to q. The computation is carried out in a number of iterations. In each iteration, two basic operations and two normalizations are executed for each page. The two operations are defined below:

   - Operation I: Update each $a(p)$ to be the sum of the current hub scores of Web pages in the base set that have a link to p. More precisely, $a(p) = \sum_{q:(q,p)\in E} h(q)$, where E is the set of links with both involved pages from the base set T.
   - Operation O: Update each $h(p)$ to be the sum of the current authority scores of Web pages in the base set that are linked from p. More precisely, $h(p) = \sum_{q:(p,q)\in E} a(q)$, where E is the same as in Operation I.

- After all authority and hub scores have been updated in the current iteration, normalize each authority score and each hub score as follows:

$$a(p) = \frac{a(p)}{\sqrt{\sum_{q \in T}[a(q)]^2}}$$

$$h(p) = \frac{h(p)}{\sqrt{\sum_{q \in T}[h(q)]^2}}$$

The above computation process is repeated until the scores converge.

It is shown in [12] that the authority and hub scores of all pages are guaranteed to converge by the above algorithm.

After all scores are computed, the Web pages in the base set are sorted in descending authority score and the pages with top authority scores are displayed to the user.

The process for retrieving n Web pages with top authority scores for a given query q can be summarized as follows.

Algorithm Search_by_Authority(q, n)
begin
submit q to a similarity-based search engine to obtain the root set S;
expand S to the base set T;
initialize authority and hub scores $a(p) = h(p) = 1$ for each page in T;
repeat for each page p in T
{ apply Operation I;
apply Operation O;
normalize $a(p)$ and $h(p)$;
}
until the scores converge;
sort pages in descending authority scores;
return the top n pages;
end;

Example 2.3 Consider the directed graph in Figure 2.3. Suppose nodes in the graph correspond to Web pages and directed edges denote links. Suppose the pages in the graph form the base set T. We now compute the hub score and authority score of each page. Let $a(p)$ and $h(p)$ denote the authority and hub scores of page p. Initially, $a(p) = h(p) = 1$ for each page p.

In the first iteration, we have $a(a) = 1, a(b) = 0, a(c) = 0, a(d) = 2, a(e) = 3, h(a) = 5, h(b) = 3, h(c) = 5, h(d) = 0, h(e) = 1$. After normalization, we have $a(a) = 0.267, a(b) = 0, a(c) = 0, a(d) = 0.535, a(e) = 0.802, h(a) = 0.645, h(b) = 0.387, h(c) = 0.645, h(d) = 0, h(e) = 0.129$.

In the second iteration, we have $a(a) = 0.129, a(b) = 0, a(c) = 0, a(d) = 1.29, a(e) = 1.678, h(a) = 2.969, h(b) = 1.678, h(c) = 2.969, h(d) = 0, h(e) = 0.129$. After normalization, we have $a(a) = 0.061, a(b) = 0, a(c) = 0, a(d) = 0.609, a(e) = 0.791, h(a) = 0.656, h(b) = 0.371, h(c) = 0.656, h(d) = 0, h(e) = 0.029$.

After 5 iterations, the following are the approximate converged scores: $a(a) = 0, a(b) = 0, a(c) = 0, a(d) = 0.615, a(e) = 0.788, h(a) = 0.657, h(b) = 0.369, h(c) = 0.657, h(d) = 0, h(e) = 0$. The

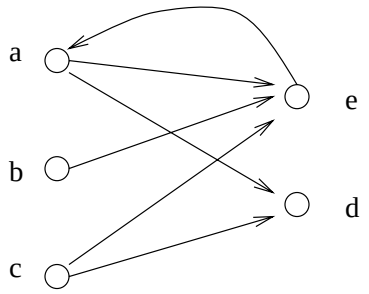

Figure 2.3: A Web Graph for Example 2.3

authority score of page e is higher than that of page d because e is pointed to by more good hub pages. It is interesting to note that the authority score of page a is 0 even though it is pointed to by page e. The reason is that e is not a good hub page as its hub score is 0. Clearly, that a is not a good authority page and that e is not a good hub page reinforce each other. ■

Both operations I and O can be expressed as matrix multiplication. Let A be the adjacency matrix of the Web graph induced by Web pages in the base set. The (i,j)-th entry of the matrix is 1 if the i-th page has a link to the j-th page and the entry is 0 if there is no such a link. A is an $n \times n$ matrix if the base set has n pages. Let A^T be the transpose of A; a_i and h_i be the vectors of authority scores and hub scores after the ith iteration, i=1, 2, ... Then the operation I in the ith iteration can be expressed as $a_i = A^T h_{i-1}$ and the operation O in the ith iteration can be expressed as $h_i = A a_i$, where h_0 is the initial vector of hub scores.

Notice that after the identification of the pages in the root set, the entire computation of authority and hub scores is based on the link structure associated with the pages in the root set. In other words, only in the step for identifying the pages in the root set, the similarities of pages with respect to the given query are taken into consideration and the remaining steps do not use information regarding document similarity.

The above method for calculating authority and hub scores treats all links the same. This is reflected in the adjacency matrix A as each link is associated with a value (weight) of 1. In reality, some links may be more useful in identifying authoritative pages with respect to a given topic that other links. Two cases are considered below.

1. Two types of links can be distinguished based on whether or not the two pages related to a link are from the same domain, where the domain name of a page is the first level string of its URL. For example, if the URL of a page is "www.cs.binghamton.edu/~meng/meng.html", then the domain name of the page is "www.cs.binghamton.edu". A link between pages with different domain names is called a *transverse link* and a link between pages with the same domain name is called an *intrinsic link*. It can be argued that transverse links are more significant than intrinsic links for two reasons. First, intrinsic links are sometimes used for presentation purposes, i.e., breaking a large document into smaller linked pieces to facilitate browsing. Second, intrinsic links can be considered to be self-referencing whose significance should be lower than references by others. At an extreme case, intrinsic links may be solely added by an author to artificially boost the authority of his/her own pages (i.e., *link spamming*). One way to handle intrinsic links is to simply discard them [12]. Another method is to give a

lower weight to intrinsic links. This can be achieved by associating a value smaller than 1 with each intrinsic link in the matrix A [3].

2. As mentioned previously, an anchor text is often associated with each link. It can be argued that if the anchor text of a link contains some terms in the query (topic), then the likelihood that the page pointed to by the link is an authoritative page with respect to the topic is increased. In general, a vicinity of a link can be defined to include terms within certain distance (say 50 characters) on both sides on the link. Then the weight associated with a link can be defined as an increasing function of the number of terms in the vicinity of the link that appear in the query. In [2], the weight formula is $w(p,q) = 1 + k(p,q)$, where (p,q) is a link from page p to page q and $k(p,q)$ is the number of terms in the vicinity of the link that appear in the query. This weight can be incorporated into the computation of authority and hub scores by replacing the (p,q) entry in matrix A by $w(p,q)$.

Multiple Communities

For a given topic, the Web may contain multiple communities that are related to the topic. Each community can be characterized as a set of Web pages that is related to a common topic and are densely linked. In comparison, there are relatively few links between different communities. There are many reasons for the existence of multiple communities related to the same topic. One possibility is that the authors of the pages in one community are not aware of the existence of other communities. Another possibility is that the communities representing competing groups or even hostile groups that do not want to link to each other. For example, pro-choice and pro-life groups are both related to abortion and their Web pages may not reference to each other. A third possibility is that the topic as defined by some terms actually has multiple meanings. For example, the communities that are related to the topic "house" may include the one that involves pages related to "house of representatives" (i.e., US Congress) and the one relating to "the sale of house in real estate market".

The existence of multiple communities related to the same topic can cause problems to the retrieval of desired pages. Specifically, if the algorithm discussed above is used to compute authority and hub scores for a given topic, then it is likely that only good authoritative pages in the largest community (i.e., the one with the largest size) can be found and smaller communities may not yield any results. In other words, the largest community will dominate smaller communities. Let us use an example to explain the reason of this phenomenon. First it can be observed that the pages in a community roughly form a bipartite graph with hub pages and authority pages (see Figure 2.4). Suppose the pages in the base set T for a given query contains two communities C1 and C2 such that size of C1 is larger than that of C2. Specifically, we consider the case when C1 has $2n$ nodes (pages) with n hub pages and n authority pages, and C2 has $2m$ nodes with m hub pages and m authority pages, $n > m$, such that every hub page has a link to every authority page in the same community but there are no links across different communities.

Suppose initially the hub and authority scores of all pages in T are 1. In the first iteration, after Operation I, all authority pages in C1 have the authority score of n and all authority pages in C2 have the authority score of m; after Operation O, all hub pages in C1 have the hub score of n^2 and all hub pages in C2 have the hub score of m^2. Now we perform normalization. Let $A_1 = \sqrt{n * n^2 + m * m^2}$ and $H_1 = \sqrt{n * n^4 + m * m^4}$. Then after normalization, for each authority page p in C1, the authority score is $a(p) = n/A_1$ and for each authority page q in C2, the authority

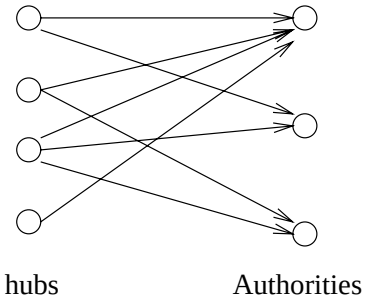

Figure 2.4: A Web Community

score is $a(q) = m/A_1$. Similarly, for each hub page p in C1, the normalized hub score is $h(p) = n^2/H_1$ and for each hub page q in C2, the normalized hub score is $h(q) = m^2/H_1$. Now consider the second iteration. After Operation I, all authority pages in C1 have the authority score of $n * n^2/H_1$ and all authority pages in C2 have the authority score of $m * m^2/H_1$; after Operation O, all hub pages in C1 have the hub score of $n^2 * n^2/H_1$ and all hub pages in C2 have the hub score of $m^2 * m^2/H_1$. Let A_2 and H_2 be the normalization factors of authority scores and hub scores for the second iteration, respectively. Then after normalization, for each authority page p in C1, the authority score is $a(p) = n^3/(H_1 * A_2)$ and for each authority page q in C2, the authority score is $a(q) = m^3/(H_1 * A_2)$. In general, it can be shown that after k iterations, the normalized authority score for each authority page in C1 is $a(p) = n^{2k-1}/(H_1 * \cdots * H_{k-1} * A_k)$ and the normalized authority score for each authority page in C2 is $a(q) = m^{2k-1}/(H_1 * \cdots * H_{k-1} * A_k)$, where H_i is the normalization factor of hub scores in the ith iteration ($H_0 = 1$) and A_k is the normalization factor of authority scores in the kth iteration. Since $n > m$, when k is sufficiently large, $a(q)/a(p) = (m/n)^{2k-1}$ will approach zero, i.e., $a(q)$ will become negligible in comparison to $a(p)$. As pages are retrieved in descending authority scores, pages from community C2 are unlikely to be retrieved.

One method that enables the retrieval of pages from multiple communities is as follows. First, Algorithm Search_by_Authority is applied against the original base set T. This enables the retrieval of good authoritative pages from the largest community in T. If the user is not satisfied by the results (i.e., the results are not from the community expected by the user) or the user simply wants to retrieve more documents from possibly smaller communities, he/she makes a request for results from a different community, say by clicking the "next community" button. Upon receiving the request, the search engine removes all pages in T whose authority scores are above certain threshold (e.g., significantly different from zero). This has the effect of removing pages from the largest community. As a result, the largest community is destroyed. Next, the same algorithm is applied against the remaining pages. This permits the retrieval of good authoritative pages from the second largest community. Obviously, this process can be repeated until either the user is satisfied or all communities have been exhausted. Alternatively, the search engine may automatically repeat the above process and display the results from different communities separately without waiting any further request from the user.

The search engine HITS (Hyperlink-Induced Topic Search) [12] retrieves Web pages in descending order of authority scores. Currently, IBM's CLEVER project [6] is to develop a new search engine that incorporates the latest research results based on the hub and authority theory.

2.3.4 Comparisons of Different Methods

In this subsection, we briefly compare the three methods of utilizing the linkage information, namely *Vector Spread Activation*, *PageRank* and *Hub and Authority*, based on the three properties associated with PageRank in Section 2.3.2.

1. From Formula (2.1), we can see that the ranking scores of pages computed by the *Vector Spread Activation* method satisfy the first two properties of the three properties. To see this, consider page p_i in Formula (2.1). When there are more links to page p_i, more link(j,i) = 1 will be true and therefore the ranking score of p_i, $rs(q, p_i)$ will become higher. Furthermore, the higher the values $rs(q, p_j)$'s are, the higher the value $rs(q, p_i)$ will be. However, Formula (2.1) does not satisfy the third property as it does not take the number of child pages of any parent page into consideration.

2. The authority scores can also be said to satisfy the first two properties among the set of pages in the same community. When more pages in the same community point to a page p, the authority score of p, namely $a(p)$ is likely to be increased. Furthermore, if p is pointed to by important (hub) pages, then $a(p)$ will also likely be increased. However, the third property is not satisfied by the authority scores as the contribution of a page, say q, to the authority score of a child page is independent of the number of child pages of q. Unlike the *PageRank* method and the *Vector Spreading Activation* method which compute a single quantity for each page, two quantities, namely hub and authority scores, are computed for each page in the *Hub and Authority* method.

3. In the *Vector Spread Activation* method, the importance of a page is dependent on a given query and is directly related to the similarities of the page and its parent pages. Unlike the other two methods, the computation the *Vector Spread Activation* method is not carried out repeatedly until the scores converge. In the *Hub and Authority* method, the authority score of a page is also dependent on a given query but is less directly related to the similarities of Web pages with respect to the query. The similarities are used to identify the root set but after the root set is identified, document similarities are no longer used in subsequent computation of authority scores. PageRanks are computed independent of any given query. However, through Formula (2.6), it is possible to combine PageRanks and similarities to retrieve important pages related to a given topic. It would be interesting to compare the retrieval effectiveness of the three methods based on a common testbed. There is currently no reported study on such a comparison.

2.4 Collaborative Filtering

When a user submits a query to a search engine, the user may have some of the following behaviors or reactions regarding the returned Web pages:

1. Click some pages in certain order while ignore others.

2. Read some clicked pages longer than some other clicked pages.

3. Save/print certain pages.

4. Follow some links in clicked pages to reach more pages.

The reaction of a user u to the results of a query q can be considered as a piece of knowledge associated with the user-query pair (u, q). The same user may use the search engine many times with different queries. Each time, the user reacts to the retrieved results. Many users may submit different queries to the search engine and have their own reactions to returned results. Thus, a vast knowledge base can be obtained based on all users' reactions to the returned results of their queries. This knowledge base is potentially very useful to improve the retrieval performance of the search engine as many users may have common information needs and the same or similar queries may be submitted by different users. A challenge is how to make use of the knowledge base to improve the effectiveness of the search engine.

There are at least three different ways the knowledge base could be utilized.

1. Use the knowledge obtained from a user's reaction to the results of his/her current query immediately to benefit the current search needs of the user. This is very similar to the relevance feedback introduced in Chapter 1 except that here the user does not explicitly tell the search engine which returned pages are relevant, instead, the search engine may deduce which pages are likely to be useful based on the user's reactions. For example, if a page is saved, printed or viewed for a significant amount of time, then the page can be considered to be relevant.

2. Use the knowledge obtained from a user's reactions to the results of his/her current query as well as that of the user's previous queries to benefit the future search needs of the user. The idea is to use the queries as well as derived relevant pages to each query to built one or more *profiles* for the user. Each user profile contains information, say appropriate terms, to indicate one interest of the user. For example, if a user has submitted queries to search Web pages related to finance and environment, then one profile for finance and one profile for environment can be constructed for the user. A sample profile for environment could contain terms such as "forest", "ozone", "pollution", "environment", etc. A profile could be constructed using important terms appearing in related queries and documents.

   User profiles can benefit a user in a number of ways. First, they can be used to determine the meaning of a query term. If a user submits a query with a single term "bank" and the user has a profile on environment but no profile on finance, then it is likely that the current usage of this term is like in "river bank" rather than in "investment bank". Second, when an appropriate profile can be identified to be closely related to a query, then terms in the profile may be added to the query (i.e., *query expansion*) such that a longer query can be processed. In text retrieval, it is known that longer queries tend to return better matched documents because they are often more precise in describing users' information needs than short queries.

3. Use the knowledge obtained from all users' reactions to the results of their current and previous queries to benefit the future search needs of every user. This is essentially an application of *collaborative filtering* in the search engine environment. According to [16], "Collaborative filtering systems make use of the reactions and opinions of people who have already seen a piece of information to make predictions about the value of that piece of information for people who have not yet seen it." For the rest of this subsection, we describe how collaborative filtering can help with text retrieval in the context of a search engine.

A collaborative filtering system is sometimes also called a *recommendation system* with the following three major components:

Recommendation gathering: This component records users' reactions to retrieved documents. Essentially it keeps track of who has found what documents to be useful and how useful with respect to what query. As discussed above, the usefulness of a document could be derived from viewing time and other user reactions.

Recommendation aggregation: This is to combine multiple recommendations into a useful measure. A simple aggregation is to count the number of times that a particular document has been recommended (i.e., considered to be useful) for a particular query. A more elaborate solution may weigh each recommendation with higher weights for stronger recommendations. In PHOAKS system [20], the measure is the number of unique recommenders.

Recommendation usage: This is to apply recommendation measures to recommend documents to retrieve for a given query. One possibility is to use a recommendation measure directly to rank documents. Another possibility is to combine the recommendation and the document similarity to rank documents.

The DirectHit search engine (www.directhit.com) is a collaborative filtering based search engine. According to [9], there are three paradigms to provide access to Web pages.

1. *Author-controlled search engines.* Most search engines on the Web retrieves pages for a given query based on the content of each page. As the content of a page is controlled by the author of the page, these search engines can be considered as author-controlled. One big problem with such type of search engines is that the quality of a page such as writing style cannot be taken into consideration. Worse yet, an author may use different ways to trick a search engine by adding popular terms into his/her page even though the terms are not related to the real content of the page (i.e., spamming. See next section for more discussion on spamming).

2. *Editor-controlled directories.* In this case, all pages that are searchable have been examined and organized into a hierarchy by some one (editor) who created the directory. As a result, the editor controls the order in which pages related to a given topic are viewed. The most famous example for this type of directory is Yahoo. This method can ensure the quality of pages to a large extent. However, this method, if carried out manually, is very time-consuming and only a very small portion of the Web can be scanned by the editor(s).

3. *User-controlled search engines.* This type of search engine relies on the collective feedback of all users to determine the quality of a page. If a page is viewed by many users with similar information needs (i.e., their queries are similar), then it is likely that the page is of good quality and should be retrieved when a related query is submitted. This approach can not only safeguard the quality of retrieved pages but also scale to a large number of Web pages. One weakness of this approach is that pages that have not been retrieved previously (such as newly added pages) will have less chance to be retrieved. One possible solution is to take the freshness of a page into consideration. For newly added or recently updated pages, the ranking function gives more weight to content-based similarity and less weight to user recommendation. As a page becomes older, more emphasis will be placed on the recommendation and less on content-similarity.

2.5 Other Issues

2.5.1 Additional Search Parameters

In addition to the contents and links associated with a page, other secondary information can be utilized as search parameters. Examples include the date in which a page is created, the latest date in which it is modified and the organization which publishes the page. These parameters may permit more accurate retrieval.

2.5.2 Dynamic Environment

In the Web environment, changes are usually made a lot more frequently than in a non-Web environment. In standard document retrieval, adding new documents is more likely than making changes. The reason is that historically, when a book or a paper is published, it is not changed, until there is a new edition. At that time, it can be classified as a new document. In the Web environment, since information is stored electronically, whenever there is a need for changes, the document is modified. Furthermore, Web pages are stored in autonomously managed Web servers. When a Web page is modified, a new page is added to a local server or an existing page is deleted from a local server, the search engine is usually not notified. As a result, search engines often have index information based on obsolete documents. One way to deal with the problem is to let the robot re-visits different sites periodically to find updated pages. As the Web is very large and growing rapidly, it takes a lot of resources to visit all the sites once. Currently, a newly added page may be added to the index of a search engine in weeks. As a consequence, there is a need to develop better methods to detect document changes so that more effort can be focused on sites that have more changed documents first.

2.5.3 Combating Spamming

Spamming refers to techniques that are employed to increase the chance of a page being retrieved by search engines even when the content of the page is not related to a given query. Spamming can have different forms. For example, popular terms that do not reflect the real contents of a page may be added into the page so that when these popular terms appear in a query the page will have a better chance to be retrieved. As an example, if there is a current hot topic, then the keywords related to that hot topic may be added to the page as spamming terms. Furthermore, the popular terms could be repeated many times in a page so that they will have high term frequencies. As term frequency weight is widely used by search engines and is an increasing function of term frequency, this increases the likelihood for the page to be retrieved. In order not to affect the presentation of the main contents of a page, spamming terms may be de-emphasized or hidden from viewers. Common tricks are to use tiny font for spamming terms and place them as the end of the pages, place spamming terms in comments so that they would not show up for viewers and put spamming terms in the color which is identical to the background color of the page so that they would not be visible to viewers. Spamming may also appear in the form of links. For example, in order to artificially increase the popularity and/or authority of a page, the author may create many dummy pages with links to the page.

A good search engine should employ retrieval techniques that can either eliminate or significantly curb the influence of spamming so that better quality documents can be returned to users. For

spamming terms, the indexer of a search engine may choose to ignore terms in comments and terms that have the same color as the background, ignore or significantly reduce the weights of, terms that are in very small fonts. More sophisticated indexers may analyze whether terms with high term frequencies appear in grammatically incorrect sentences as spamming terms often repeat many times by themselves and do not form correct sentences. By placing significant weight on linkage-derived importance as in the PageRank approach and the authority approach, we can reduce relying on content-based similarity. This can also curb the impact of spamming terms. For spamming links, the search engine may ignore intrinsic links or give them low significance. In the PageRank approach, the significance of a link is tied to the importance of the page containing it. As pages housing spamming links are unlikely to be important, the influence of spamming links can be curbed.

Bibliography

[1] S. Brin, and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. WWW7 Conference, 1998.

[2] S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, P. Raghavan, and S. Rajagopalan. Automatic Resource Compilation by Analyzing Hyperlink Structure and Associated Text. 7th International World Wide Web Conference, 1998.

[3] S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, S. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Mining the Web's Link Structure. IEEE Computer, August 1999.

[4] Fah-Chun Cheong. Internet Agents: Spiders, Wanderers, Brokers, and Bots. New Riders.

[5] J. Cho, H. Garcia-Molina, and L. Page. Efficient Crawling Through URL Ordering. 7th WWW Conference, 1998.

[6] http://www.almaden.ibm.com/cs/k53/clever.html.

[7] M. Cutler, Y. Shih, and W. Meng. Using the Structures of HTML Documents to Improve Retrieval. USENIX Symposium on Internet Technologies and Systems (NSITS'97), Monterey, California, 1997, pp.241-251.

[8] M. Cutler, H. Deng, S. Manicaan, and W. Meng. A New Study on Using HTML Structures to Improve Retrieval. Eleventh IEEE Conference on Tools with Artificial Intelligence (ICTAI'99), Chicago, November 1999.

[9] The Direct Hit Popularity Engine Technology: A White Paper (http://www.directhit.com/about/products/technology_whitepaper.html).

[10] T. Haveliwala. Efficient Computation of PageRank. Technical Report, Stanford University, 1999.

[11] B. Janson, A. Spink, J. Bateman, and T. Saracevic. Real Life Information Retrieval: A Study of User Queries on the Web. ACM SIGIR Forum, 32:1, 1998.

[12] J. Kleinberg. Authoritative sources in a hyperlinked environment. 9th ACM-SIAM Symposium on Discrete Algorithms, 1998.

[13] S. Lawrence, and C. Lee Giles. Accessibility of Information on the Web. Nature, 400, July 1999, pp. 107-109.

[14] M. Mauldin. Lycos: Design Choices in An Internet Search Service. IEEE Expert Online, February 1997.

[15] O. McBryan. GENVL and WWWW: Tools for Training the Web. First WWW Conference, Geneva, May 1994.

[16] B. Miller, J. Riedl, and J. Monstan. Experiences with GroupLens: making Usenet Useful Again. Proceedings of the 1997 Usenix Winter Technical Conference, January 1997.

[17] R. Motwani, and P. Raghavan. Randomized Algorithms. Cambridge University Press, United Kingdom, 1995.

[18] L. Page, S. Brin, R. Motwani, and Terry Winograd. The PageRank Citation Ranking: Bring Order to the Web. Technical Report, Stanford University, 1998.

[19] J. Rennie, A. McCallum. Using Reinforcement Learning to Spider the Web Efficiently. International Conference on Machine Learning, 1999.

[20] L. Terveen, W. Hill, B. Amento, D. McDonald, and J. Creter. PHOAKS: A System for Sharing Recommendations. The Communications of the ACM, 40:3, March 1997, pp. 59-62.

[21] B. Yuwono, and D. Lee. Search and Ranking Algorithms for Locating Resources on the World Wide Web. IEEE Conference on Data Engineering, pp.164-177, 1996.