The CLEVER search system

Ravi Kumar* Prabhakar Raghavan† Sridhar Rajagopalan* Andrew Tomkins*

Abstract

This chapter describes the CLEVER search system developed at the IBM Almaden Research Center. We present a detailed and unified exposition of the various algorithmic components that make up the system, and then present results from two user studies.

1 Introduction

The subject of this chapter is the CLEVER search system developed at the IBM Almaden Research Center. Our principal focus is a detailed and unified exposition of the various algorithmic components that make up the system. Many of these have hitherto appeared in a number of papers and reports; some have appeared in incomplete form; and others have never been disclosed. In addition, we summarize the results of two user studies performed during the project.

The web has proven to be a fertile test bed for combining ideas from human behavior and social network analysis together with traditional information retrieval. The latter discipline has focused on relatively focused corpora that are small, with uniform and high-quality documents. The networking revolution made it possible for hundreds of millions of individuals to create, share and consume content on a truly global scale, demanding new techniques for information management and searching. One particularly fruitful direction has emerged here: exploiting the link structure of the web to deliver better search, classification and "mining". The idea is to tap into the annotation implicit in the actions of content creators: for instance when a page-creator incorporates hyperlinks to the home pages of several football teams, it suggests that the page-creator is a football fan.

But the large body of work leveraging this link structure has developed a unifying theme which we echo here: hyperlinking provides a powerful and effective tool, but only in conjunction with other content-based techniques. The Google search engine [10], for instance, makes heavy use of link analysis in the form of PageRank values [3]; however, the ranking algorithm has roughly a hundred additional terms that make the approach truly effective. Similarly, the CLEVER search system began with purely link-based techniques, and over time incorporated a variety of content-based additions, many in response to specific conditions prevailing on the web.

Historically, many of these ideas first came into existence in the CLEVER project with the development of Kleinberg's HITS algorithm [11] at IBM, during 1997. The framework of HITS was both elegant and extensible, leaving open a number of avenues for further exploration and refinement. Many research groups seized on (and continue to explore) these possibilities (see Section 2 for more background); the CLEVER project at IBM was perhaps the earliest of these. From 1997 to 1999, the project focused largely on search, before shifting to a focus on measurement and mining. Our goal here is to provide a unified and complete view of the salient technical results derived from the project in the search domain.

The remainder of the paper proceeds as follows. In Section 2 we present some background in the area of link analysis. However, in order to keep the presentation focused, we assume that the reader has some familiarity with the domain. Section 3 describes the

*IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120.

†Verity Inc, 892 Ross Dr., Sunnyvale, CA 94089. This work was done while the author was at the IBM Almaden Research Center.

CLEVER search system in detail. Section 4 then presents some experimental results, and Section 5 concludes.

2 Background

Link analysis [11, 3], particularly in conjunction with text-based methods [10, 2, 5, 22], is generally believed to enable significant improvements in ranking quality. However, implementing link-based ranking methods for the WWW is challenging, primarily due to the highly variable nature of web pages.

The basis for the CLEVER system is the HITS [11] algorithm. We describe it now for completeness. The interested reader is referred to the original paper for greater detail. Given a collection of pages, the HITS algorithm computes two scores corresponding to each page in the collection. The first is the *hub* score: this measures the value of the page as a collection of resources. The second is the *authority* score: which measures public opinion of the quality of any page. The two scores are expressed as vectors ($\vec{h}$ and $\vec{a}$) with as many dimensions as the number of pages in the collection, and with each dimension corresponding to a page.

Kleinberg posits that hub pages and authority pages of good quality share a mutually reinforcing relationship, namely, good hub pages point to many good authorities, and good authority pages are pointed to by many high quality hub pages. Letting A be the citation matrix, i.e. $A_{ij} = 1$ if and only if page i points to page j, we can mathematically state this relationship using the formulae:

$$\vec{a} \propto A^T \vec{h}, \qquad \vec{h} \propto A\vec{a}$$

and furthermore, since both $\vec{h}$ and $\vec{a}$ are scoring functions, we have $\vec{h}, \vec{a} \geq 0$. Here A^T is the transpose of matrix A. Under fairly general conditions on A (see [9] for details), the simultaneous equations above have a unique non-negative, non-zero, solution. The hub-vector $\vec{h}$ is the principal eigenvector of AA^T, and the authority vector $\vec{a}$ is likewise the principal eigenvector of $A^T A$. Both vectors can be computed approximately by iterative methods, or exactly by Gaussian elimination.

A number of systems and algorithms have been proposed as extensions to the HITS framework, including the Salsa system of Lempel et al [14], and the work of Toyoda and Kitsuregawa (see for instance [21]). We focus here on the CLEVER system in particular, and refer the interested reader to the citations above for more information about related approaches.

CLEVER represents a set of two classes of extensions within the HITS framework. The first class consists of modifications that preserve the overall structure of the HITS iterations, but allows the matrix to contain real values rather than simply 0 or 1 values. Thus, clever uses edge weights to reflect how relevant each link is to the subject of the query. The principle here (sometimes known as lexical affinities [15]) is that if relevant text occurs in the proximity of the link, the link is more likely to be significant. The exact computation of the weight matrix is somewhat more complex and is described in Section 3.4.

The second class of modifications comprises extensions to the HITS framework itself; these modifications often fall into the category of heuristics, and are discussed throughout Section 3.

It is instructive to compare HITS to PageRank[3], the ranking system used as a basis for the Google search engine[10]. Consider the matrix M where $M_{ij} = 1/d_i$. Here d_i is the out-degree of page i, and specifies the exact number of out-links on it. This matrix is also known as the Markov matrix related to the graph defined by A. Unlike the HITS family of algorithms, the PageRank algorithm computes the the eigenvector of $\lambda M + (1-\lambda)[1/n]$. Here n is the number of pages (nodes) in the entire graph, and $[1/n]$ represents the matrix with each entry $1/n$. The motive for the PageRank algorithm comes from the "random surfer model." Imagine a random surfer, who follows links out of a page at random with probability λ and once in a while (with probability $1-\lambda$) jumps to an entirely new random page on the web distributed uniformly among all pages. The principal eigenvector of $\lambda M + (1-\lambda)1$ measures how often such a surfer would visit each page i if he were to continue browsing for an infinitely long time. Brin and Page (see [3]) posit that this quantity can be used as a static estimate of page quality.

We digress at this time to a recent observation of [12]. The "random jump" with probability λ gives the PageRank iteration a greater stability, in that page ranks tend not to change dramatically with the deletion of a few edges. HITS and related schemes are conversely less stable. If HITS (and variants) were to be "relaxed" using a similar linear combination, then this effect can be mostly mitigated. We refer the interested reader to [12].

Similarly, the question of stability raises the web question of resilience to *link spamming*, or creation of pages whose out-links are chosen to increase the score of some target in a link-based ranking scheme. PageRank requires that important pages be cited by important pages, and therefore incorporates natural resistance to link spam. However, systems exist to spam both HITS and PageRank, and any well-engineered system will require safeguards against such behavior.

Finally, we should note that the implementation of PageRank in the Google system is query-independent; the PageRank scores provide a static ranking of all web pages, and this ranking is used with many other factors to produce the final output for a particular query. The CLEVER system, on the other hand, incorporates the query directly into the matrix at runtime. Therefore, implementing the runtime is a greater challenge. In particular, we cannot consider computing the CLEVER rank for every page on the web given a query.

Instead, we implement CLEVER as a three stage process. The first stage is a naive indexing step in which we simply choose a small number of pages (the root set) which contain the query terms. In the second stage, we do a small focussed crawl starting from the root set to identify a larger collection of pages to process (the base set). Finally, we rank all the pages in the base set using the algorithms defined in Section 3. This brings us to a fine distinction in the implementation described in this paper and those which are more common in web scale search engines. Our algorithms are best viewed as refinement algorithms, those which given a small (or medium) sized collection of pages, refines the search within this set. In our context, two pages may be ranked in different orders given different queries.

3 The CLEVER system

Section 2 presents Kleinberg's HITS algorithm, which represents a clean and mathematically-grounded framework. However, as discussed earlier, there are a number of issues that arise in building an actual system to implement this algorithm, and in modifying the algorithm to deal with the vicissitudes of web data. Addressing these issues resulted in the CLEVER system. In this section we describe the system and give some motivation for the particular extensions we found necessary.

The scope of this paper does not allow us to provide a "Users Manual" of the system—instead, we seek to provide a sense of the implementation, and enough algorithmic details that the reader could reproduce the the functionality.

We begin in Section 3.1 with a description of some of the concepts behind extensions to the HITS framework. Next, Section 3.2 describes some system issues such as the hardware for which the system has been developed, and the parameters and control files that influence the functioning of the system. Then Section 3.3 describes the data gathering, or crawling, phrase of the algorithm. Section 3.4 describes the process of constructing a weighted graph based on the crawled data. Section 3.5 then describes the actual iterative procedure applied to the graph. Finally, Section 3.6 describes the final creation of the output.

3.1 Extensions to the HITS Framework

In this section we list a number of components of CLEVER search which represent extensions to (or in some cases departures from) the traditional HITS framework. We also seek to give some motivation for the extensions, based on real-world examples of structures that might lead a naive link-based algorithm astray.

- *Controlling the Engine:* Execution of the CLEVER system depends on the values of a number of parameters (currently 57). These parameters are specified in a set of SGML-style configuration files, and can in many cases be set

or modified using an advanced web search front-end.

- *The Query Language:* The system allows users to specify five different sets of keywords. The first set is sent to the search engines in order to seed the search, but is not used to weight links between pages. The second set is used for link weighting, but does not influence the queries sent to search engines. The third set, which is used by default for keywords entered from a search front-end, influences both the search engine query and the link weighting algorithm. The fourth set contains terms that a page must include in order to be part of the final output. And the fifth and final set contains terms that a page may not include in order to be part of the final output. Typical users arriving through the web front-end use only the third set; more sophisticated users may use all five.

  All of these sets are specified using keyword search semantics based on Altavista's "Advanced Search" language. Query terms may be single words or double-quoted phrases. Terms may be preceded by a + or −, to imply emphasis or de-emphasis of the term. These modifiers impact crawling because they are sent directly to the search engines used to generate the initial set of pages. They also impact the graph generation stage because keyword modifiers have significant influence on the weights assigned to links, based on keywords that are roughly proximate to the anchor.

- *Incorporation of Textual Content into Edge Weights:* There are a number of factors used to determine the weight of a particular edge (see Section 3.4 for details), but the most important is the textual content surrounding the location of the anchor. If the anchortext contains the query term, the edge should be treated as highly relevant. While the anchortext itself is critical, we also allow text appearing within some window of the anchor to influence the weighting, with a diminishing contribution depending on distance from the anchor. We refer to the graph showing contribution as a function of word location as the "tower bridge function": flat along the anchortext itself, then falling off linearly on either side.

  More formally, let a_{ij} be zero if there is no link from page i to page j, and be positive if a link exists. The value of the textual contribution to that edge weight has a contribution from each query term. The contribution of a query term appearing at distance i within a window W of terms from the hyperlink is proportional to $W - i$. Query terms within quotes are treated as atomic units, so the word "car" generates no contribution for the query "vintage car."

- *Nepotistic Links:* A *nepotistic link* has both source and destination on the same web site. Such links represent a form of *self-promotion*, in that a web site confers authority upon itself. Thus, we seek to discard these links. It therefore becomes important to determine when two pages are on the same site. To make this determination, we use information about the URL and IP address of the two pages. See Section 3.4 for details.

- *Covering Heuristic:* The value of a hub page is by definition in its links rather than its contents. If all the destinations accessible from a particular hub are also accessible from better hubs, we do not need to output this hub. More generally, we seek to return a set of hub pages that together contain as many unique, high-quality links as possible. We therefore apply a well-known greedy set-cover heuristic as follows. Once the iteration step has converged, we repeat the following process until we have generated the correct number of hubs: return the best hub, zero the authority values of all pages pointed to by the hub, and recompute hub values. See Section 3.6 for details.

- *Authority Packing Heuristic:* Despite the removal of nepotistic links, it is possible for instance for an organization's homepage, and several children of that page, to accumulate authority. However, in the final output we wish to pro-

vide the user with as much authoritative substance as possible in a small number of pages. To achieve this, after each step of the iteration we "re-pack" the authority of any site, as follows: if multiple documents within a logical site (as defined above) have non-zero authority, the authorities of all but the page with the largest authority are set to zero. See Section 3.5 for details.

- *Hub Functions:* Many resource compilations (e.g., bookmark files) contain pages pertinent to a number of disjoint topics. This causes such compilations to become good hubs, which in turn causes irrelevant links from the same page to become good authorities. To address this problem we note that pointers to pages on the same topic tend to be clustered together in resource compilations. We allow each link in a web page to have its own hub value, so the hub value of a page is now a function of the particular link rather than a constant. When computing authority values, the authority of the destination is incremented by the hub value of the link. When recomputing hub values, the authority value of the destination is used to increment the hub value of the source link, and according to a spreading function, the hub values of neighboring links. Thus, useful regions of a large hub page can be identified. The final hub value of a page is the integral of the hub values of its links. See Section 3.5 for details.

- *Page relevance:* A set of irrelevant but highly-linked pages might accidentally creep into the crawled set of pages, and become highly-ranked hubs and authorities due to their mutual reinforcement. Or more likely, some pages might be clearly "on-topic," while others might be of limited or zero relevance. The algorithm makes strong use of text near links in order to determine relevance of a link, but nonetheless, we would prefer not to conflate pages that are entirely irrelevant with pages that are relevant but happen not to have a keyword appearing near a link.

  Thus, we use a traditional ranking function to determine the overall relevance of the page to the query. Based on the relevance of the source and destination pages, we increase or decrease the weight of links between them. See Section 3.4 for details.

- *Exemplary Pages:* The user may identify certain URLs as *exemplary hubs, authorities, or sites*. Exemplary hubs are high-quality collections of links to topical pages. Exemplary authorities are high-quality pages about the topic. Exemplary sites are both. Additionally, the user may identify *stopsites*: URLs that should not be crawled under any circumstances. See Sections 3.3 and 3.4 for a full description.

  These types of pages impact the algorithm in two ways. First, they influence the set of pages crawled by the system. Second, they influence the edge weights connecting hyperlinked pages. The following describes these issues in more detail:

  - Each exemplary hub is added to the graph along with its out-neighbors, and all pages connected (through in-links or out-links) to the hub or its out-neighbors.
  - Each exemplary authority is added to the graph along with all pages that point to at least two exemplary authorities, and all pages connected (through in-links or out-links) to the authority such an in-neighbor.
  - Each exemplary site is added to the graph with its in- and out-neighbors, and all pages connected (through in-links or out-links) to the page or its neighbors.
  - Each stopsite is eliminated before being crawled.

  The intuition behind these rules is the following. We believe the pages pointed-to by an exemplary hub to be high quality candidate authorities. Therefore, pages that point to these pages have a better than average chance of being good hubs. So we add these candidate to the graph. Similarly for exemplary authorities:

pages that point to two or more exemplary authorities are high-quality candidate hubs, so we place them in the initial set so any candidate authorities they point to will be added to the root set. The asymmetry in treatment of exemplary hubs and exemplary authorities arises because the user who specifies an exemplary hub knows all the out-links of the page, while the user who specifies an exemplary authority may not know all the in-links of the page.

Exemplary sites also influence edge weighting. Intuitively, the edges that point to example authorities or edges originating at example hubs should weigh more. Additionally, if a page is cited in the *lexical neighborhood* of example authorities, then that link should weigh more. Let $w(x, y)$ denote the weight of the edge from x to y in the graph. The following four heuristics are in addition to the basic edge-weighting schemes stated in [4, 5]: (1) If x is an example hub and x points to y, then $w(x, y)$ is increased; (2) If y is an example authority and x points to y then $w(x, y)$ is increased; (3) If y is an example authority and x points to both y and y' in the same lexical neighborhood, then $w(x, y')$ is increased; and (4) If y and z are example authorities, and x points to y' in the same lexical neighborhood with both y and z and the reference to y' is between the references to y and z then $w(x, y')$ is increased.

Consider searching for long-distance phone companies. If Sprint and AT&T are example authorities for this node, and both occur in a single list of links, we have strong evidence that the other elements of the list may be relevant to the topic. However if the list contains only AT&T then we have only weak evidence that the list is about long-distance phone companies. The increase in weight of an edge is a super-linear function of the number of links to example authorities occurring the edge, and of the proximity of the edge to these links.

3.2 System issues

The code for CLEVER is written in C and C++, and runs under Linux (RedHat 6.2) and AIX. It represents approximately 10K lines of code. The system is invoked from the command line, and leaves its results in various support files. It is also invoked through a set of cgi scripts which we do not describe. The entire system resides on a single machine, and does not support distributed operation. It executes as a single process and fetches data using multiple threads.

Various parameters (currently 57, as noted earlier) control the operation of the system. In the initial phases of the algorithm, these parameters are loaded from a global default SGML-style config file, a similar user-specified config file, and the command line. If the query has been run recently and the result is cached, then the system returns the cached result and exits. The setup phase then checks for various consistency conditions such as the presence of data directories, and exits with an error if any of these tests fail.

In the following sections, various numerical parameters control the behavior of the algorithm. For concreteness, we often specify default values for these parameters. The defaults typically result from some exploration of the parameter space, but typically have not been subjected to a rigorous sensitivity analysis.

3.3 Crawling

The CLEVER system performs five phases of crawling, after which the system has resident a set of web pages relevant to the query upon which link and text analysis can be performed. Each of the five phases uses the same crawling infrastructure. We begin with an overview of that infrastructure, and then describe the five phases in turn.

3.3.1 Operation of the Crawling Module

The crawling module operates as follows. First, a Crawl object is created to control the fetching threads. This object reads all URLs into memory, and creates job records for them. It guarantees that no stopsites are fetched, and that no page

is fetched more than once at any point during processing. The Crawl object then creates a number of fetcher threads, based on a system parameter. As the fetcher threads retrieve pages, the Crawl object monitors the total number of pages successfully fetched. Based on a profile depending on the fraction of pages retrieved, the fetch is aborted if a certain amount of time passes from completion of the most recent fetch.

Each fetching thread regularly checks a shutdown flag to determine whether it should terminate. If not, it retrieves the next job from a shared data structure, and spawns a fetch. The fetch takes a timeout parameter which is passed down to all socket operations. The fetcher then operates as follows. First, it checks to see that the maximum number of redirects has not yet been reached for this URL. Next, it verifies the hostname and port. It opens a connection either to the server, or to a SOCKS server if necessary, sends the http request, and waits for data with a timeout. As data arrives, it reads it into a buffer with a maximum allowable data size. If the result is a redirection, it changes the target URL and iterates.

At the conclusion of a fetch operation, the crawling module returns the status of the fetch.

We found that following redirects is essential to gathering a sufficiently high quality dataset. Further, we found that a realistic approach to timeouts if crawling is being performed in response to a user query, is also essential. Fetching a few hundred or thousand pages could take a few seconds to a few minutes, with reasonable timeout procedures, but the last 10% of the pages might take substantially longer and might never be successfully retrieved.

Similarly, like many before us, we found that DNS code is not well suited for rapid crawling in a single process, and that the reentrant versions are either buggy, or lock a process-global mutex, or both. We modified the DNS fetch code in order to allow multithreaded operation.

3.3.2 Phases of Crawling

Crawling Phase 1: Search Engine Queries

The crawling is seeded by querying 200 pages from internet search engines. The formats of six search engines are known to the system, and a control parameter specifies the subset to be used. The 200 pages are split evenly between all search engines specified in the control parameter.

The search engine query is built by gathering all relevant keywords (see the Query Language discussion in Section 3.1), possibly including new keywords generated by an aliasing mechanism that allows a user to specify a "concept" which has been associated with a full sub-query. This query is then sent to engine-specific generators which create URLs for the fetches. To this resulting set of query URLs, the system also adds the URLs for all exemplary sites and exemplary hubs, and adds URLs that will fetch from a search engine the in-links to all exemplary sites and exemplary authorities. As a result, the rootset will contain exemplary hubs and their out-links, and exemplary authorities and their in-links. After expansion of the rootset, as described in Section 3.1, the final graph will contain pages that link to the sorts of pages linked-to by exemplary hubs, and also pages that are pointed-to by the kinds of pages that point to exemplary authorities.

These query URLs are then sent to the crawling module. The resulting pages are parsed and their out-links are gathered into a set called the rootset_urlfiles. All exemplary pages (including exemplary hubs, authorities and sites) are then added to the rootset_urlfiles. Finally, additional URLs may be specified via a parameter to be added directly into rootset_urlfiles.

Next, each URL in rootset_urlfiles is passed to an in-link query generator that creates a URL to be sent to a search engine that will extract in-links to the page. These in-link query URLs are all written to a set called rootfile_inlink_queries to be fetched during the third phase of crawling.

During the parsing of the query URLs and the pages of the rootset, a number of limits are imposed on the number of links a single page can generate, and the number of links the entire set can generate. Once a per-page limit is reached, processing of the page is aborted. Once a per-set limit is reached, processing is aborted for the entire set.

Crawling Phase 2: The Rootfiles

Next the rootset_urlfiles are sent to the crawling

module. The resulting pages are parsed, and the extracted URLs are written to a set called rootfile_outlinks. These are fetched during phase 4.

Crawling Phrase 3: Querying In-links to Rootfiles

Next the rootfile_inlink_queries are crawled. These URLs contain queries to search engines, built to extract pages that link to rootfile URLs. The resulting search engine pages are parsed, and links are extracted into a set called rootfile_inlinks. These are fetched during phase 5.

Crawling Phrase 4: Rootfile Out-links

Next, the rootfile_outlinks set is sent to the crawling module, which results in all the pages linked-to by pages in the rootset.

Crawling Phrase 5: Rootfile In-links

Finally, the rootfile_inlinks set is crawled, resulting in pages that link to pages in the rootset.

3.4 Graph Generation

Now that crawling is complete, all further processing is local to the machine. This section describes the creation of a weighted graph representing a combination of link and text information. The following section then describes the iterative algorithm to process that graph in order to generate final hubs and authorities.

Initially, the system gathers together all query terms to be used in edge weighting, and breaks them into positive (terms with a + modifier), negative (terms with a − modifier), and unsigned (all other terms). Query term matching for edge weighting is case-insensitive, so all terms are downcased. All pages are then scanned for occurrences of query terms. The following naive ranking function is used to determine the relevance of a page to the set of query terms, for use in the page relevance heuristic described in Section 3.1. For each page, the number of positive, negative, and unsigned query terms is computed. Pages are then split into three class, based on relevance: strong, normal, and weak. Any page containing a negative query term, or containing no query terms, is automatically weak. Otherwise, let p be the number of positive terms in the query. Then a page is strong if it contains at least two distinct query terms, and also contains at least $\min(2, p)$ distinct positive terms. Otherwise, a page is normal.

3.4.1 Filter Terms

As described in the Query Language overview of Section 3.1, there are two sets of parameter keywords that contain terms which must, or must not, be contained on any page that becomes part of the final output of the system. These keyword sets are referred to as "postfilters." Postfilters do not affect the execution of the algorithm, but pages that fail some postfilter are not output. The "include postfilter" consists of a set of terms, any of which may have a "+" modifier. To pass the filter, a page must contain every "+" term, and at least one of the other terms in the filter, if any. The "exclude postfilter" consists of a set of unmodified terms. To pass this filter, the page must not contain any term in the set. The results of this filtering set are used during output.

3.4.2 Graph Vertex Construction

A "shingle" value is computed for each page along the lines of the computation defined by Broder [19], and duplicates are removed. Within a class of duplicates, one page at random is chosen as the representative of that class and kept. All edges whose destination is a duplicate page are modified to point instead to the primary representative of their shingle class. In addition, the following pages are removed:

- pages whose URL is a search engine
- pages that are too small (10 bytes or fewer)
- stopsites (different rules for intranet and internet)

Next, titles are generated for each page, to be used during output. Also, the URL of each page is canonicalized, and IP addresses are gathered for all pages (this information was computed during the crawling phase). The canonical form of the URL and the IP address are used below to determine whether two pages are deemed to be from the same site.

3.4.3 Graph Edge Construction

Each page is now parsed, and all hyperlinks are extracted. If the destination is not in the graph, the edge is removed from consideration. Further, a parameter gives a hard limit on the total number of out-links from a page. Additionally, relative URLs are not added to the graph (except in processing intranet data). Finally, "nepotistic edges" within a site are not considered.

The Same Site Algorithm Two pages are deemed to be from the same site if they meet either of the two following conditions:

1. They have similar IP addresses, determined as follows:

Address Class	Match Requirements
Class A and B addresses	two top octets
Class C addresses	three top octets
Class D addresses	all octets

2. They have similar URLs, based on the following three rules:
   (a) URLs of the form ".../∼joe" or ".../[Uu]sers/joe" are treated as being from site "...joe"
   (b) URLs that match a set of templates (ie, www.geocities.com/Colosseum/Arena/5400/, or members.tripod.com/username) are matched according to special rules
   (c) Otherwise, sites are extracted per RFC 2396.

3.4.4 Edge Weight Computation

Each page is divided into regions using the regular expression "⟨h[1-6]r⟩" as the region separator.

The weight of an edge is then determined according to a variety of local and global factors.

Local Edge Weighting Factors

1. Each edge begins with some constant weight, by default 3.
2. Fix a particular edge. A query term appearing within i terms of the anchor for that edge is given weight 10-i as described in Section 3.1. That weight is doubled for positive query terms, and negated for negative query terms. These contributions are summed over each query term occurring within 10 terms of the anchor in either direction.
3. Edges from exemplary hubs are scaled by some factor (default 1.1) and edges to an exemplary authority are scaled by some factor (also default 1.1)
4. Page relevance weighting (strong, normal, weak) is incorporated as follows. Let e denote the page relevance ranking parameter, from 0 to 100. Let s and w be the number of strong and weak pages in the set {src,dest}. Then the resulting edge weight is multiplied by the page relevance multiplier $m = 1.4^{(s-w)e/100}$
5. For each region of size < 25, the weight of each edge in the region is increased by 1.1 (resp. 1.5) if there is one (resp. more than one) exemplary authority linked-to from this region.

Global Edge Weighting Factors

1. Following Bharat and Henzinger [2], for each link between sites A and B, if there are a total of n links between A and B, the weight of each edge between the two sites is multiplied by $(1/n)^{(f/100)}$ where f is the "inter-site linking factor", from [0..100].

3.5 Iteration

Now that the graph has been computed, the iterative algorithm proceeds as follows. First, an authority score is attached to each vertex, and as described in the hub functions section above, a hub score is attached to each edge. The iteration then proceeds by repeatedly computing hub scores and authority scores, and then renormalizing. We now describe these three steps, beginning with the recomputation of authority scores as it is most straightforward. Let $h(e)$ be the hub score associated with edge e, $w(e)$ be the weight of edge e, and $a(p)$ be the authority score of page p.

recompute_auth_scores: Let $I(P)$ be the set of inlinks to page P. We set $a(P) = \sum_{e \in I(P)} h(e) \cdot w(e)$.

recompute_hub_scores: This function implements hub functions as described earlier. The algorithm processes each edge e from page P to page Q as follows. Consider the location of e within P. Define $N(e)$ to be the set of all edges that are both within the region of e within P, and within 8 edges of e in that region. During processing of e, the hub score of each edge $e' \in N(e)$ will be updated.

The "raw score" $r(e)$ is set to $a(Q) \cdot w(e)$. For each $e' \in N(e)$, let $d(e, e')$ be the number of edges between e and e' in the appropriate region of P; this is an integer between 0 and 8. We would like to add $r(e)/(1 + d(e, e'))$ to the hub score of e', but this may allow authority to be propagated down a low-weight edge, then back out a high-weight edge, causing a low authority on $A1$ to result in a huge authority score for $A2$ simply because the edge pointing to $A1$ has much lower weight. Thus, we define the edge ratio $\sigma(e, e') = w(e')/w(e)$. Finally, we can state the complete algorithm. First, $\forall e \in E, e \leftarrow 0$. Then:
$\forall e \in E, e' \in N(e)$:
$h(e') \leftarrow h(e') + r(e)\sigma(e, e')/(1 + d(e, e'))$.

renormalize: Renormalization has two steps. First, and only if the appropriate boolean parameter is set, the value of each non-maximal authority on a site is set to 0. Second, the authority and hub score vectors are normalized to have 2-norm 1.

Iteration proceeds for a fixed number of steps, usually 5-10, and then terminates.

3.6 Output

First, as many authorities as necessary are output in order from the list of all pages sorted by final authority score. Pages are output only if they pass the postfilters as described above. Once a sufficient number of authorities has been output, output of hubs proceeds as follows.

1. The hub score of each page is set to the sum of the hub scores of all edges on the page.
2. The best hub to pass the postfilter is output.
3. The authority scores of all authorities pointed-to by this hub are reduced by a constant factor between 0 and 1, given by a parameter, by default 1.
4. All hub scores are recomputed according to recompute_hub_scores().
5. All hubs are resorted, and the procedure iterates until an appropriate number of hubs have been output

4 Experiments and results

This section reports on experiments to evaluate the performance of the CLEVER search system as a search tool (Section 4.1) and as a tool for automatically constructing taxonomies (Section 4.2).

Traditional IR systems are evaluated using the measures of precision and recall on a large pre-evaluated corpus [18]. Evaluating the performance of a web search system, however, is a tricky issue for the following reasons:

- The web is large: The web contains more than 2 billion documents. At this magnitude, rating the entire web (even automatically) is out of question.
- The web is growing: Around ten million new pages are created every day. Even if one were to create a pre-evaluated web corpus, this could be used to evaluate actual search engines for only a brief window (probably shorter than the time to gather the relevance judgments) before the corpus became "stale".
- The web is dynamic: The composition of a "typical" web document in terms of links, text, graphics, etc. is changing. Therefore, labeling today's web as a corpus and using it to evaluate/compare search systems can be dangerous as the results

for today's web may not generalize to the future web.

- Search engines are incomplete: No search engine can index "all" the web. So, the notion of recall is problematic in web search.

The closest approximations to "relevance judgments" on today's web are portals such as Yahoo! and OpenDirectory, which through human involvement collect high-quality pages on a number of topics. While the above reasons imply that these portals cannot index the web exhaustively, they do provide "soundness" judgments. More precisely, for a fixed topic, if a search engine returns a page that is also indexed by the portal under that topic, then it is a strong indication that the page is of high quality; if, however (as is more likely), the portal does not index the page, we have no information about the quality of the page.

4.1 CLEVER as a search engine

We study the performance of CLEVER as a search engine. Since at the time of this study there were no standard benchmarks for evaluating web search systems[1], we compared ourselves against the then best-known automatic search engine, Altavista[1], and the then best-known human-compiled resource site, Yahoo![23]. We chose 26 broad-topic queries: +Thailand +tourism, +recycling +cans, "Gulf war", "affirmative action", "amusement park", "classical guitar", "computer vision", "field hockey", "graphic design", "lyme disease", "mutual funds", "parallel architecture", "rock climbing", "stamp collecting", "table tennis", "vintage car", HIV, alcoholism, bicycling, blues, cheese, cruises, gardening, shakespeare, sushi, and telecommuting. For these queries, we computed the precision of all three sources on a fixed number of pages according to our user-provided relevance judgments and compare these results. We refer to this technique as *comparative precision*. The details of this experiment appeared in [5].

For each of the 26 queries, we extracted ten pages from each of our three sources. Altavista and CLEVER were both given the query as it appears in the table (i.e., with quotes, plus-signs, and capitalization intact). The same search was entered manually into Yahoo!'s search engine, and of the resulting leaf nodes, the one best matching the query was picked by hand. If the best match contained too few links, the process was repeated to generate additional links. Using this procedure we took the top ten pages from Altavista, the top five hubs and five authorities returned by CLEVER, and a random ten pages from the most relevant node or nodes of Yahoo![2]. We then interleaved these three sets and sorted the resulting approximately 30 pages alphabetically (there are almost never duplicate pages from the three sources). We asked each user to rank each of these pages "bad," "fair," "good," or "fantastic" based on *how useful the page would be in learning about the query*. We took good and fantastic pages to be relevant, and then computed precision in the traditional manner. Since our users evaluated only the pages returned from our three sources, but did not know which source returned which page, we refer to this type of data as *blind post-facto* relevance judgments.

The subjective evaluation of relevance was performed by a set of 37 subjects, yielding 1369 data points. The subject was free to browse the list of pages at leisure, visiting each page as many times as desired, before deciding on a final quality score. We now outline the results of this experiment.

Precision measures. Table 1 shows the average comparative precision of each search engine over the set of 26 queries; recall that we took "good" and "fantastic" to be relevant. CLEVER outperformed both Yahoo! and Altavista under this metric. While the favorable comparison to Altavista was expected, the advantage over Yahoo! was surprising. If we consider the fraction of queries on which each search engine performed best, we find that in 50% of all topics, CLEVER was the best in terms of precision and in 31% of all topics, it tied for first place with Yahoo! Yahoo! was better than CLEVER for 19% of the topics.

[1]This situation has changed with the development of WebTrack at TREC.

[2]Yahoo! lists pages alphabetically and performs no ranking, hence the requirement that we take ten pages at random.

Measure	Yahoo!	Altavista	CLEVER
Average Precision	.38	.18	.48
Fantastic Fraction	.13	.04	.15
Linear Measure	.42	.27	.50

Table 1: Precision ratings, by search engine.

Table 1 also gives two alternate measures of overall quality. The first measure, "fantastic fraction," is the fraction of pages returned that are rated as "fantastic" (rather than either "good" or "fantastic" in our original measure). The second, "linear measure," weights a "bad" page at 0, a "fair" page at .33, a "good" page at .66 and a "fantastic" page at 1. CLEVER performs better than all other systems under all measures, although Yahoo! finds roughly as many "fantastic" pages (which is not surprising).

Precision vs. rank. We now consider the rank assigned to a page by each engine. Figure 1 plots the average precision of the top i pages for each engine, for $i = 1 \ldots 10$. For this purpose, the ranking function that we use for CLEVER interleaves the hubs and authorities starting with the best hub.

One possible concern is that a large Yahoo! node may contain many good pages and some excellent ones. Choosing only ten pages at random from such a node may penalize Yahoo! for gathering more information on the topic. However, almost all our Yahoo! nodes contained fewer than 30 pages and the correlation of precision to Yahoo! node size is minimal, only -0.09. This indicates that the concern is not serious.

Hubs vs. authorities. The precision scores of hubs and authorities show only a mild correlation (0.36). For some topics, hubs dominate, and for other topics, authorities dominate, suggesting that users find value in both types of pages. Overall, CLEVER is better at identifying hubs than authorities—in 72% of the queries, the comparative precision of the hubs was at least as high as the authorities.

It remains unclear how to judge the response set of a search engine as a whole, rather than page-by-page. Both the covering and the packing heuristics (Section 3.1) may reject pages that are individually highly rated in favor of pages that contribute to the overall quality of the response set. Hence we believe that the quality of our result set as a collection of pages will be better than the average precision metric indicates.

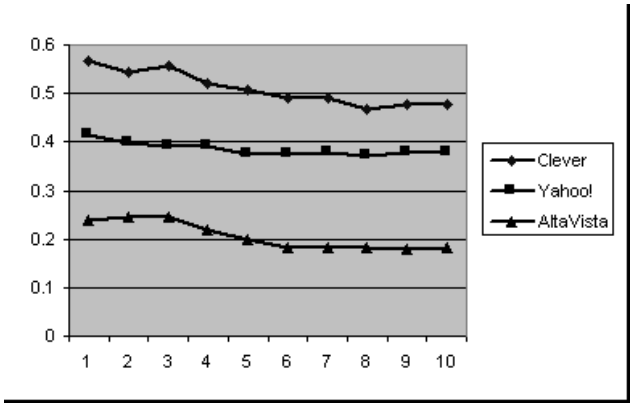

Figure 1: Precision as a function of the rank of pages.

4.2 CLEVER for automated taxonomy construction

In this section, we evaluate the performance of CLEVER as an automatic taxonomy building tool. This experiment involved a team of four ontologists (the authors of this chapter). The details of this experiment appeared in [22] and we paraphrase this below.

Our experiment involved the construction of four taxonomies. Three were drawn from predefined subtrees of Yahoo!: Government, Recreation & Sports, and Science. The fourth "personal" taxonomy consisted of nodes of personal interest to one of our ontologists. There were between 100 and 150 nodes in each of the first three taxonomies, and 70 in the personal taxonomy, for a total of 455 nodes. We built each node in the taxonomy three times: (1) Using a "naive" query consisting essentially of the topic title, with (occasionally) some simple alternatives. The intent was to simulate a near-automatic process that

gives a quick first cut at describing a node. (2) Using an "advanced text" query consist of descriptive terms and example terms. The intent was to simulate a richer text query possibly using some domain knowledge, much as in a commercial rule-based classifier. (3) Using one or more exemplary hubs and authorities (see Section 3.1 for details about exemplary pages). This is the richest form of description in our experiment—a combination of text and example sites.

Our goal in designing these experiments was to benchmark each mode of taxonomy construction, monitoring: (1) wall clock time elapsed during the construction of the taxonomy; (2) quality of resources found by each; (3) level of exemplification; (4) investment in looking at results of text searches. Our system was configured to log all the actions of our ontologists—these logs yield, among other things, the wall clock time used in taxonomy construction, the sequence of mouse clicks, the number of result pages viewed, etc.

We collected user statistics evaluating the pages as follows. We enlisted 50 users willing to help in the evaluation of our results, and decided *a priori* that each user could reasonably be expected to evaluate around 40 URL's. Therefore, we needed to spread these 2000 total URL evaluations carefully across the well over 50,000 URL's contained in our taxonomy. We adopted a random sampling approach as follows. First, we constructed the entire taxonomy in each of the three modes of operation. After all three versions of the taxonomy were constructed, we randomly sampled 200 nodes for evaluation, chosen uniformly from all nodes. Thus each user would evaluate 4 topic nodes on average; given the 40-URL limit on user patience, this suggests that each user can be expected to view 10 URL's per topic node.

CLEVER returns 25 hubs and 25 authorities for each topic node in each of the three modes of taxonomy creation, for a total of 150 URL's. Since we wish to ask each user to evaluate a total of around 10, we sub-sampled as follows. For a particular ordered list of URL's, we refer to the "index" of a particular URL to mean its position in the list—the first URL has index one, and so forth. Consider a topic node N. We chose a "high-scoring" index $h(N)$ uniformly from the indices between 1 and 3, and a "low-scoring" index $l(N)$ uniformly from the indices between 4 and 25. We then extracted the two hub (resp. authority) pages at indices $h(N)$ and $l(N)$ in the list of hubs (resp. authorities), from the taxonomy constructed using naive queries. This resulted in four URL's. We performed the same extraction for topic node N in the advanced text and example modes of creation as well, resulting in a total of 12 URL's. These samples contained some overlaps however; in all the mean number of distinct URL's extracted per node was about 10.2. From classical statistics, the score we compute is an unbiased estimator of the actual scores (cf. [8]).

We then asked each user to evaluate four topic nodes from our 200, chosen randomly without replacement. Note that, as in Section 4.1, we do not tell our users whether a particular URL was generated as a good hub or as a good authority. The evaluation methodology also followed Section 4.1, with additional ranking options of "unranked" (the initial value), and "unreachable." Pages ranked "unranked" (presumably because a user simply forgot to rank this page) or "unreachable", were not considered in the ranking. All other pages were assigned scores as follows: "bad" = 0, "fair" = 1, "good" = 2, "fantastic" = 3. When we refer to scores in the following, we mean these values. As before, when dealing with precision, we define pages ranked "good" or "fantastic" to be relevant.

Results and conclusion. Table 2 shows the average values over the top 25 results, broken down by mode of creation as well as taxonomy, in both the average score and the precision metrics. The first conclusion, shown via our user study and the timing results of our instrumented taxonomy creation tool, is that an ontologist armed with the paradigm of iterative topic creation using increasingly sophisticated forms of query can create a high-quality taxonomy with a fairly quick turnaround time. The second high-level conclusion is that the well-known benefits of relevance feedback appear to hold in the domain of hyperlinked document search. As a tertiary conclusion, we show that, at least in the context of taxon-

omy creation, the traditional "advanced query" syntax used by search engines does not provide significantly better results than more naive queries. This might provide partial explanation for user dissatisfaction with "advanced search" functions in most search engines.

An examination of the nodes shows that topics in the personal taxonomy tend to be narrower in focus. For instance, some of the nodes are `FOCS/STOC, SIGMOD, WWW, Collaborative Filtering, Latent Semantic Indexing, Phrase Extraction, Kerberos, Smartcards`. There are far fewer pages about, for instance, the FOCS/STOC (theory) conferences than about the sport of ice hockey. Interestingly, in this focused context we see the largest difference between modes: exemplification improved performance by approximately 33% over the purely textual approaches.

5 Conclusion

In this chapter, we have given a detailed description of the CLEVER search system. The system includes a broad set of extensions to the underlying HITS framework in which it was developed. We motivated and described these extensions in detail, and then provided results from two user studies to show that the resulting system gives high-quality results on real-world web ranking problems. From these results, we draw two conclusions. First, link-based ranking schemes can provide dramatic improvements over purely content-based techniques for web data. And second, such link-based schemes are most effective when augmented with textual information.

References

[1] The Altavista Search Engine, http://www.altavista.com

[2] K. Bharat and M.R. Henzinger. Improved algorithms for topic distillation in hypertext environments, *Proc. 21st ACM SIGIR, 1998.*

[3] S. Brin and L. Page. The anatomy of a large scale hypertextual Web search engine, *Proc. 7th WWW*, 1998.

[4] S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, P. Raghavan, and S. Rajagopalan. Automatic resource compilation by analyzing hyperlink structure and associated text, *Proc. 7th WWW*, 1998.

[5] S. Chakrabarti, B. Dom, Ravi Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Experiments in topic distillation. *Proc. ACM SIGIR Workshop on Hypertext Information Retrieval*, pages 13–21, 1998.

[6] J. Dean and M. Henzinger. Finding related pages on the Web, *Proc. 8th WWW*, 1999.

[7] E. Efthimiadis. *Interactive Query expansion and Relevance Feedback for Document Retrieval Systems*. Ph. D. Thesis, City University, London, UK, 1992.

[8] W. Feller. *An Introduction to Probability Theory and its Applications*. John-Wiley, 1968.

[9] C. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 1989.

[10] The Google Search Engine, http://www.google.com

[11] J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46, 1998.

[12] A. Ng, A. Zheng, and M. Jordan. Stable Algorithms for Link Analysis. *Proc. ACM SIGIR*, 2001.

[13] J. Koenemann. Supporting interactive information retrieval through relevance feedback, *Proc. ACM SIGCHI*, 1996.

[14] R. Lempel and S. Moran. The stochastic approach for link-structure analysis (SALSA) and the TKC effect. *Proc. 9th WWW*, 2000.

[15] Y. Maarek and F. Smadja. Full Text Indexing Based on Lexical Relations. *Proc. ACM SIGIR*, 1989.

Taxonomy	Advanced	Exemplary	Naive		Advanced		Exemplary	
	secs.	secs.	Avg. Score	Prec.	Avg. Score	Prec.	Avg. Score	Prec.
Science	108.0	119.8	1.61	0.55	1.53	0.52	1.63	0.56
Recreation	192.4	239.6	1.64	0.61	1.68	0.64	1.70	0.63
Personal	157.5	214.0	1.03	0.30	0.91	0.31	1.41	0.48
Government	270.4	222.4	1.45	0.51	1.44	0.50	1.42	0.48

Table 2: Average construction time per node and average score, precision of top 25 hubs and authorities, by taxonomy.

[16] J. Pitkow and P. Pirolli. Life, death, and lawfulness on the electronic frontier. *Proc. ACM SIGCHI*, 1997.

[17] P. Pirolli, J. Pitkow, and R. Rao. Silk from a sow's ear: Extracting usable structures from the Web. *Proc. ACM SIGCHI*, 1996.

[18] G. Salton and C. Buckley. Improving retrieval performance by relevance feedback. *JASIS*, 41(4):288–297, 1990.

[19] A. Broder, S. Glassman, M. Manasse, and G. Zweig. Syntactic Clustering of the Web. *Proc. 6th WWW*, 1997.

[20] C.J. van Rijsbergen. *Information Retrieval*. Butterworths, 1979.

[21] M. Toyoda and M. Kitsuregawa. A Web Community Chart for Navigating Related Communities. *Proc. 10th WWW*, 2001.

[22] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. On Semi-Automated Web Taxonomy Construction. *Proc. ACM WEBDB*, 2001.

[23] Yahoo! http://www.yahoo.com