Distributed Hypertext Resource Discovery Through Examples

Soumen Chakrabarti[1]
IIT Bombay
soumen@cse.iitb.ernet.in

Martin H. van den Berg[2]
FX Palo Alto Laboratory
vdberg@pal.xerox.com

Byron E. Dom
IBM Almaden
dom@almaden.ibm.com

Abstract

We describe the architecture of a hypertext resource discovery system using a relational database. Such a system can answer questions that combine page contents, meta-data, and hyperlink structure in powerful ways, such as "find the number of links from an environmental protection page to a page about oil and natural gas over the last year." A key problem in populating the database in such a system is to discover web resources related to the topics involved in such queries. We argue that that a keyword-based "find similar" search based on a giant all-purpose crawler is neither necessary nor adequate for resource discovery. Instead we exploit the properties that pages tend to cite pages with related topics, and given that a page u cites a page about a desired topic, it is very likely that u cites additional desirable pages. We exploit these properties by using a crawler controlled by two hypertext mining programs: (1) a *classifier* that evaluates the relevance of a region of the web to the user's interest (2) a *distiller* that evaluates a page as an access point for a large neighborhood of relevant pages. Our implementation uses IBM's Universal Database, not only for robust data storage, but also for integrating the computations of the classifier and distiller into the database. This results in significant increase in I/O efficiency: a factor of ten for the classifier and a factor of three for the distiller. In addition, ad-hoc SQL queries can be used to monitor the crawler, and dynamically change crawling strategies. We report on experiments to establish that our system is efficient, effective, and robust.

1 Introduction

There is a growing need for future generations of hypertext search engines that transcend keyword-based search and permit powerful query and discovery by combining predicates on page content, page and hyperlink meta-data, and hyperlink graph structure. Several projects including TSIMMIS[3] [19], WebSQL[4] [26], W3QS[5] [24], WEBL[6] [22], STARTS[7], WHIRL[8], and generalized proximity search [18] have made important advances in the data representation and query optimization for semi-structured and unstructured domains.

Our goal in the Focus project is to go beyond representation and basic querying, to discover properties that combine the topical content of web pages and the linkage relationship between them. We give some examples of such advanced query power:

Citation sociology: Find a topic (other than bicycling) within one link of bicycling pages that is much more frequent than on the web at large. The answer found by the system described in this paper is *first aid.*

Typed link: Find computer science theory faculty who have advised at least one student who has published on database topics.

Spam filter: Find pages that are apparently about database research which are cited by at least two pages about Hawaiian vacations.

Community evolution: Find the number of links from a page about environmental protection to a page related to oil and natural gas over the last year.

The novelty in the examples above is that page content is selected by topics, not keyword matches. In the examples above, the topics are expressed syntactically using simple phrases, but we envisage that in actual use, our system will not depend on the keyword 'bicycling' to select bicycling related pages from the web (many of which might not contain that word) but instead learn to identify bicycling related pages from a set of examples provided by the user. Our goal is to answer such queries while materializing as little of the distributed hypertext repository (in general, the Web) as possible.

In the spirit of Mendelzon and Milo [27], we claim that the traditional self-contained view of a database is not appropriate for answering such questions. The queries above involve small fractions of the web. A standard crawler would waste resources and yet likely to provide stale and incomplete results; the biggest search engines cover 35–40% of the Web today [3].

[1]Work partly done at IBM Almaden
[2]Work done at IBM Almaden

Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, 1999.

[3]http://www-db.stanford.edu/tsimmis/tsimmis.html
[4]http://www.cs.toronto.edu/~websql/
[5]http://www.cs.technion.ac.il/~konop/w3qs.html
[6]http://www.research.digital.com/SRC/WebL
[7]http://www-db.stanford.edu/~gravano/starts.html
[8]http://whirl.research.att.com/

Thus, a large crawl of the web, followed by filtering based on topic (used earlier in a search engine setting [12]) is a poor solution.

This paper deals with the implementation of a novel example-driven, goal-directed data acquisition system. Its goal is to answer a pre-specified set of standing queries continually, not answer interactive queries unless they are contained in the topics crawled to answer the standing queries. Keeping the access costs of the web 'database' in mind, we propose the following problem formulation.

1.1 Problem formulation

We are given a directed hypertext graph G. G is distributed, as is the case for the web, and there is a non-trivial cost for visiting any vertex (web page) of G. There is also a tree-shaped hierarchical topic directory C such as Yahoo!. Each topic node $c \in C$ refers to some pages in G as examples (provided manually). We denote the examples associated with topic c as $D(c)$. These pages can be preprocessed as desired by the system. The user's interest is characterized by a subset of topics $C^* \subset C$ that is marked `good`. No good topic is an ancestor of another good topic. Topics in the subtree of a good topic are called *subsumed* topics. Ancestors of good topics are called `path` topics.

Given a web page q, a measure of relevance $R_{C^*}(q)$ of q w.r.t. C^*, together with a method for computing it, must be specified to the system. C^* will be omitted if clear from the context. In this paper, we will use a probability measure $0 \leq R(q) \leq 1$. By definition, $R_{\{\text{root}\}}(q) = 1 \forall q$. If $\{c_i\}$ are children of c_0, then $\sum_{c_i} R_{c_i}(q) = R_{c_0}(q)$.

The system starts by visiting all pages in $D(C^*)$. In each step, the system can inspect its current set V of visited pages and then choose to visit an unvisited page from the crawl frontier, corresponding to a hyperlink on one or more visited pages[9]. Informally, the goal is to visit as many relevant pages and as few irrelevant pages as possible, i.e., to maximize average relevance. Therefore we seek to find $V \supseteq D(C^*)$ where V is reachable from $D(C^*)$ such that $\sum_V R(v)/|V|$ is maximized.

1.2 Rationale and discussion

The above formulation is primarily for clarity. When G is the web, there is no hope of designing an optimal algorithm, or evaluating practical algorithms against the optimal visited set. Indeed, part of the paper (§3) deals with this non-trivial evaluation problem, measuring various indirect indicators of effectiveness. W.r.t. almost any meaningful C^*, the average relevance of the whole web is extremely small, therefore we expect $|V| \ll |G|$. Thus the system is naturally prevented from visiting and pre-processing the entire web graph, as is attempted by current web crawlers. The user's choice of C^* is a clean way to capture the recall-precision trade-off. Crawlers with very general topics in C^* (such as the root of the topic tree) have large recall whereas topic nodes deep in the topic tree induce high precision.

In a self-contained database, there are, in principle, simple formulations for query by example. There is a universe of n objects U, and given a probe or query object q and a distance measure d, the goal is to find the top $k \ll n$ objects u with the smallest values of $d(q, u)$. Query by Image Content (QBIC) is an example of this paradigm [29].

There are a number of reasons why this model is inadequate for our purposes. The biggest problem is data acquisition. E.g., it is unreasonable to have to acquire and analyze 350 million web pages to decide upon the thirty pages most relevant to the user's interest. Another serious problem is the definition of similarity or relevance. Fairly successful representations and similarity measures have been devised for images, but keyword matches are known to be relatively unreliable for searching text. Third, the diverse styles of hypertext authoring, together with the abundance of pages relevant to broad topics, makes it difficult to completely identify the hyperlinked community from a keyword query response [31].

1.3 Contributions

Our main contribution is the design of a novel example-driven, goal-directed resource discovery system. Our system uses two basic properties of the web: pages cite pages on related topics, and a page that points to one page with a desired topic is more likely than a random page to point to other pages with desired topics. Accordingly, we use two devices: a supervised *classifier* that learns the topic(s) of interest from the user's examples, and then guides the crawler accordingly, and a *distiller*, which runs concurrently with the crawler, identifying nodes in the growing crawl graph that are likely to lead to a large number of relevant pages. (Such pages are not necessarily very relevant in themselves.)

Another contribution is the implementation of the system on a relational database. The three modules, crawler, classifier, and distiller, together with monitoring and administering utilities, run concurrently as clients. The database is not merely a robust data repository, but takes an *active role* in the computations involved in resource discovery (§2.1.3 and §2.2.3). An important aspect of this work is the design of flexible schemes for crawl frontier management, and the representation of the classifier and distiller in a relational framework.

We give experimental evidence that our system is effective at discovering topic-specific web subgraphs. In particular, relevant and high-quality pages are found

[9]But also see §3.2 for more general notions of page visitation.

several links away from the results of keyword searches, establishing the superiority of our approach. We also show how a careful DBMS implementation yields better I/O performance in the classifier and distiller.

1.4 Related work

Although hypertext classification [8] and hyperlink-based popularity analysis (PageRank [5], HITS [23], CLEVER [7] and topic distillation [4]) and similarity search [15] have been studied before, no notion of adaptive goal-directed exploration is evident in these systems. Another important distinction of our system is the integration of topical content into the link graph model. PageRank has no notion of page content[10], and HITS and CLEVER explore the web to a preset radius (typically, 1) from the keyword query response. All involve pre-crawling and indexing the web.

A few systems that gather specialized content have been very successful. Cho et al compare several crawl ordering schemes based on link degree, perceived prestige, and keyword matches on the Stanford University web [13]. Ahoy![11] is a homepage search service based on a crawler specially tuned to locate homepages. Cora[12] is a search engine for computer science research papers, based on a crawler trained to extract such papers from a given list of starting points at suitable department and universities. Information filtering agents such as WebWatcher [20], HotList and ColdList [30], Fish Search [16], Shark Search and Fetuccino [2], and clan search [32] have served a similar purpose. These are special cases of the general example- and topic-driven automatic web exploration that we undertake.

2 Architecture

Our problem formulation in the previous section does not in itself suggest a procedure to attain that goal. If pages of all topics were finely dispersed throughout the web, there would be little hope for finding coherent communities. This, however, is not the case. Most citations are made to semantically related pages. Two rules can be exploited:

Radius-1 rule: Compared to an irrelevant page, a relevant page is more likely to cite another relevant page.

Radius-2 rule: The unconditional probability of a random web page u pointing to a page of a given topic may be quite small. However, if we are told that u does point to one page v of a given topic, this significantly inflates the probability that u has a link to another page of the same topic.

[10]Although Google likely combines PageRank with content-based heuristics.

[11]`http://www.cs.washington.edu/research/ahoy`

[12]`http://www.cora.jprc.com/`

These claims have been demonstrated using corpora such as patents from the US Patent Office and web pages cataloged in Yahoo!. E.g., a page that points to a given first level topic of Yahoo! has about a 45% chance of having another link to the same topic.

In this section we will describe how to exploit these properties. The radius-1 rule is exploited by a **classifier** which makes relevance judgments on pages crawled to decide on link expansion. The radius-2 rule is exploited by a **distiller** which identifies pages with many links to unvisited promising links. The system is built around a **crawler** with dynamically reconfigurable priority controls which are governed by the classifier and distiller.

2.1 Classification

The relevance of each URL fetched is evaluated using a *classifier*. Many different methods for text classification have been studied [1, 6, 14, 17]. Here we will use a Bayesian classifier. This broad family of classifiers have been found to be computationally very inexpensive and yet quite effective for high-dimensional applications such as text. The CMU *World Wide Knowledge Base* project has often used simple Bayesian learning algorithms[13] with good results. Silicon Graphics MineSet includes a Bayesian classifier[14].

2.1.1 Classifier computations

To simplify the exposition, we will propose a simple statistical generative model for documents which has been found to be surprisingly effective. Let classes be denoted c in formulae and `cid` in SQL code. For each term t (from a universe of terms) and each class c, there is a parameter $\theta(c, t)$. (The value of this parameter is unknown and is estimated from the training documents). Terms are denoted as `tid` in formulae. Given numeric values of all θ, a document d is generated as follows. First, a class or topic is chosen using a *prior* distribution. All documents by definition belong to the root of the topic tree; $\Pr[\text{root}] = 1$. Given that we have decided to write a document about internal topic node c_0, we refine the decision by picking a child of c_0: class c_i is picked with probability $\Pr[c_i|c_0]$, whose logarithm[15] is denoted $\texttt{logprior}(c_i)$ later.

Once the (leaf) class node c has been decided, the length of the document is set arbitrarily. (One may also pick the length using a class-conditional distribution, but we keep the model simple.) Having picked the length $n(d)$, we write out the document term after term. Each term is picked by flipping a die with as

[13]`http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-11/www/wwkb/`

[14]`http://www.sgi.com/software/mineset/mineset_data.html`

[15]We work in the log domain because the absolute probability is often very small.

many sides as there are terms in the universe. The face corresponding to term t has probability $\theta(c,t)$ of turning up. In the resulting document, the number of occurrences of t is denoted $n(d,t)$ in formulae and `freq`(d,t) in SQL. The class-conditional probability of generating the document is $\binom{n(d)}{\{n(d,t)\}}\prod_{t\in d}\theta(c,t)^{n(d,t)}$. This is called the Bernoulli or multinomial model for text generation.

During the setup stage, the classifier is *trained* with example documents associated with each node or class in the taxonomy tree. Training is in some sense the opposite of generation: given documents with associated classes, the classifier constructs statistical models for each class and stores these on disk. At each internal node c_0 in the class tree, training involves three steps:

Feature selection: Of all the terms in the universe, a subset $F(c_0)$ is selected. Intuitively, these are terms that provide the maximum *discrimination power* between documents belonging to different subtrees of c_0. Because training data is limited and noisy, accuracy may in fact be *reduced* by including more terms. Feature selection has been studied in detail elsewhere [6].

Parameter estimation: For all internal classes c_0 and all $t \in F(c_0)$, we have to estimate $\theta(c_i,t)$ for each child c_i of c_0 from the training documents. Let $D(c_i)$ be the set of training documents for class c_i. Then $\theta(c_i,t)$ is estimated as:

$$\frac{1+\sum_{d\in D(c_i)} n(d,t)}{\left|\cup_{d\in D(c_0)}\{t\in d\}\right| + \sum_{d\in D(c_i)}\sum_{t\in d} n(d,t)} \qquad (1)$$

Notice that $\theta(c_i,t) > 0$ for all c_i and t, but for most t, $n(d,t) = 0$ (text data is very sparse). To avoid losing the sparseness, we only store `logtheta`$(c_i,t) = \log\theta(c_i,t)$, where $\sum_{d\in D(c_i)} n(d,t) > 0$, and `logdenom`$(c_i) = \log\Big(\left|\cup_{d\in D(c_0)}\{t\in d\}\right| + \sum_{d\in D(c_i)}\sum_{t\in d} n(d,t)\Big)$.

Index construction: In traditional classification, the final indexing table, called `BLOB` here, is built as a map from (c_0,t), where $t \in F(c_0)$, to a set of records. Each record is for a child c_i of c_0 (all children need not be present), of the form $(c_i,$ `logtheta`$(c_i,t))$. This structure is shown in Figure 1. c_0 is denoted `pcid` and c_i are denoted `kcid`. There is also a `TAXONOMY` table mapping c_i to `logdenom`(c_i) and `logprior`(c_i).

During crawling, each page is subjected to *testing*. The traditional purpose of testing is to assign the document to one or few best-matching classes. A test document d is routed as follows. Suppose the root is denoted c_0. In test mode, d is tokenized, and for each term t an index probe is made with the key (c_0,t). For those $t \in F(c_0)$, a set of c and θ values are retrieved. These are used to update the probability that d was generated from each child of c_0, given it was generated from c_0 (this can be applied recursively from the root using chain rule):

$$\begin{aligned}\Pr[c_i|c_0,d] &= \Pr[d,c_i,c_0]/\Pr[d,c_0]\\ &= \Pr[d,c_i]/\Pr[d,c_0]\\ &= \Pr[c_i|c_0](\Pr[d|c_i]/\Pr[d|c_0]), \qquad (2)\end{aligned}$$

because $d \in c_i \Rightarrow d \in c_0$. (See **SingleProbe** in Figure 2 for details.)

The newly evaluated nodes c_i are checked into a pool of nodes to be further expanded. From this pool, the node with highest probability is picked as the new "root" and the above process is repeated. Typically, one may stop after evaluating the highest probability leaf node.

2.1.2 Modifications for resource discovery

In the administration phase, the user has marked a set of classes as `good`. A simple strategy for resource discovery could be the hard focus rule.

Hard focus rule: Suppose the best leaf class if the current page d is determined to be c^*. If some ancestor of c^* is `good`, insert the outlinks in d into the crawl frontier.

We won't go into details, but this turns out not to be a good rule; crawls controlled by this rule may *stagnate*, i.e., stop because the entire crawl frontier is found unsuitable for expansion. Manual inspection typically shows that the frontier nodes are quite relevant, but the best *leaf* class is not a descendant of a good class.

Soft focus rule: For queries on the materialized Web subgraph, the hard focus notion may be used to select relevant pages. For data acquisition, however, we are better off evaluating for each document d, the probability that it is good,

$$R(d) \equiv \texttt{relevance}(d) = \sum_{\texttt{good}(c)} \Pr[c|d], \qquad (3)$$

and prioritizing page crawls (partly) based on this number. We will report only on the soft focus rule because it is more robust.

2.1.3 I/O efficient implementation

We will first describe the common implementation of the classification, explain why this has poor performance, and give a superior implementation coded directly in SQL. Performance analysis is deferred to §3.

Traditionally, keyword-based near-neighbor search is done using the *inverted file* approach. For classification, we do not need to search for documents by keywords. We can therefore use a more compact inverted file representation, described next.

Keyword indices are constructed by assigning unique ID's to each term (we use t in formulae, `tid` in SQL) and each document (d in formulae, `did` in SQL).

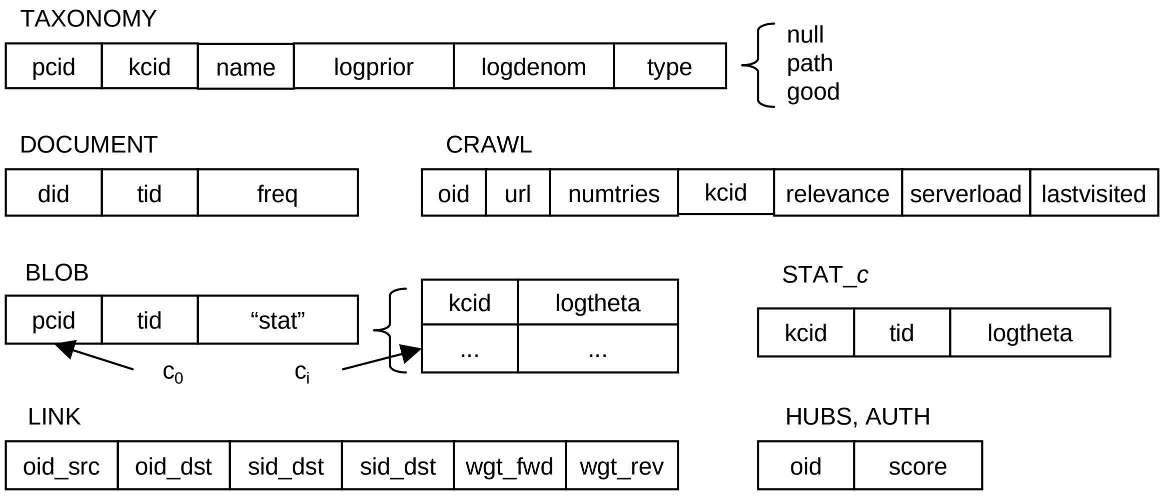


Figure 1: The main DB2 tables used in our system. `TAXONOMY`, `STAT`, `BLOB`, and `DOCUMENT` are used by the classifier. `CRAWL` and `LINK` are used by the crawler. `LINK`, `HUBS` and `AUTH` are used by the distiller. Columns which are not self-explanatory are explained in the text.

In our system we use 32-bit hash codes for terms. For topics (also called classes, denoted c in formulae and `cid` in SQL) we use 16-bit ID's.

Consider internal node c_0 with children $\{c_i\}$. Let $t \in F(c_0)$ be a feature term w.r.t. c_0 (see §2.1 for terminology and notation). The compact inverted file maps (c_0, t) to a sparse vector of θ values, each element being of the form $(c_i, \texttt{logtheta}(c_i, t))$. There is an entry for c_i only if $\sum_{d \in D(c_i)} n(d, t) > 0$. A separate table stores a map from c_i to $\texttt{logdenom}(c_i)$. $\theta(c_i, t)$ can be reconstructed from these numbers.

A key step in classification is to compute $\log \Pr[c_i | c_0, d] = \texttt{logprior}(c_i) + \log \Pr[d|c_i] - \log \Pr[d|c_0]$, where $\log \Pr[d|c_i] = \text{constant} + \sum_{t \in d \cap F(c_0)} \texttt{freq}(d,t) \texttt{logtheta}(c_i, t)$. Note that for a pre-specified C^*, we need to compute $\Pr[c|d]$ only for $c \in C^*$.

The pseudocode using the `BLOB` table (shown in Figure 1) is shown in Figure 2. With regard to disk access, it is similar to any standard keyword indexing engine, and is quite insensitive to the exact math. If, for example, the term distribution is changed from multinomial to Gaussian, θ would be replaced by the mean μ and variance σ, but not much else would change.

For large taxonomies θ cannot be stored entirely in memory. E.g., models derived from about 2100 nodes of Yahoo! and 1.5 GB of text occupy about 350 MB. (As users refine and personalize the taxonomy and ask more queries, this size gets larger.) Consequently most of the classification time is spent in the **PROBE** step. Even with caching, there is little locality of access, because the records are small and most storage managers use page-level caching. A lot of random I/O results, making the classifier disk-bound. The random I/O problem is especially serious in our system because of the multi-threaded crawler. In experimental runs, about thirty threads fetch a total of 5–10 pages a second, a typical web page having 200-500 terms, each term leading to a **PROBE**.

We will now describe a way to classify a large batch of documents using a sort-merge technique, which can be written, with some effort, directly in SQL. The tables needed are shown in Figure 1. The `TAXONOMY` table encodes the relation between parent and child classes in the usual way. Some topics are marked `good` as described earlier; all their ancestors are marked `path`. Nodes marked `null` are not of interest in a particular crawl (but these may be marked otherwise for a different crawl). The `DOCUMENT` table consists of rows of the form $(d, t, \texttt{freq}(d, t))$. Since (d, t) is a key we will refer to this table as $\texttt{freq}(d, t)$. For each internal node c_0 of the taxonomy there is a table $\texttt{STAT\_}c_0$, which stores the map from (c_i, t) to $\texttt{logtheta}(c_i, t)$, where c_i are children of c_0 and only those $t \in F(c_0)$ appear in the table. Note that only `DOCUMENT` has to be populated at crawl time; the rest is precomputed. Populating `DOCUMENT` is part of standard keyword indexing anyway.

Going through the steps in Figure 2, we note that the main step is to evaluate $\sum_{t \in d \cap F(c_0) \cap c_i} \texttt{freq}(d,t) \texttt{logtheta}(c_i, t) - \sum_{t \in d \cap F(c_0), t \notin c_i} \texttt{freq}(d, t) \texttt{logdenom}(c_i)$.

The first sum is an inner join, but the second sum leads to random update I/O. The whole expression is best rewritten (after some trial and error) using one

```
SingleProbe($c_0, d$)
For all children $\{c_i\}$ of $c_0$
    initialize array of log-probabilities $\{L[i]\}$
For each term $t \in d$ occurring $\texttt{freq}(d,t)$ times:
    PROBE BLOB with key $(c_0, t)$
    If $t \notin F(c_0)$ skip $t$
    For each $(c_i, \texttt{logtheta}(c_i, t))$ retrieved:
        $L[i] \leftarrow L[i] + \texttt{freq}(d,t)\texttt{logtheta}(c_i,t)$
    For each $c_i'$ that was missing
        $L[i] \leftarrow L[i] - \texttt{freq}(d,t)\texttt{logdenom}(c_i')$
Normalize $\vec{L}$ so that $\sum_i e^{L[i]} = 1$
For each $i$
    Assign $L[i] \leftarrow L[i] + \log\Pr[c_0|d] + \texttt{logprior}(c_i)$
```

Figure 2: Document-by-document classification pseudocode.

```
BulkProbe(c0)
with
   PARTIAL(did, cid, lpr1) as
   (select did, TAXONOMY.kcid,
      sum(freq * (logtheta + logdenom))
      from STAT_c0, DOCUMENT, TAXONOMY
      where TAXONOMY.pcid = c0
      and STAT_c0.tid = DOCUMENT.tid
      and STAT_c0.kcid = TAXONOMY.kcid
      group by did, TAXONOMY.kcid)
   DOCLEN(did, len) as
   (select did, sum(freq) from DOCUMENT
      where tid in (select tid from STAT_c0)
      group by did),
   COMPLETE(did, kcid, lpr2) as
   (select did, kcid, - len * logdenom
      from DOCLEN, TAXONOMY where pcid = c0)
select C.did, C.cid, lpr2 + coalesce(lpr1, 0)
from COMPLETE as C left outer join PARTIAL as P
on C.did = P.did and C.cid = P.cid
```

Figure 3: Hierarchical bulk classification expressed as a ODBC/JDBC subroutine. This is repeatedly called at all path nodes in topological order to evaluate the score of the good nodes.

inner and one left outer join [11]:

$$\sum_{t \in d \cap F(c_0) \cap c_i} \texttt{freq}(d,t)\Big(\texttt{logtheta}(c_i,t) + \texttt{logdenom}(c_i)\Big) - \texttt{logdenom}(c_i) \sum_{t \in d \cap F(c_0)} \texttt{freq}(d,t).$$

Figure 3 shows a high-level pseudocode for an ODBC/JDBC routine with one parameter c_0 which indicates the node at which bulk evaluation is desired. For simplicity, the code ignores some details such as priors and normalization which are lower order performance concerns.

2.2 Distillation

The purpose of distillation is to identify *hubs*, i.e, pages with large lists of links to relevant resources. A very relevant page without links is only a finishing point in the crawl. In contrast, hubs are good for crawling, and good hubs should be checked frequently for new resource links.

2.2.1 Query-based distillation review

The Web is an example of a *social network*. The edges of a social network can be analyzed to identify pages that are 'central' in some sense [21, 28, 33]. Similar techniques have been applied to the Web graph. Brin and Page [5] model the 'prestige' of a page v as roughly the sum total of the prestige of pages that cite v. If $\mathbf{E}$ is the node adjacency matrix, the prestige of a node is the appropriate component of the dominant eigenvector of $\mathbf{E}$. Kleinberg offers a slightly different model: each node v has both a *hub score* $h(v)$ and an *authority score* $a(v)$; these are estimated by mutual recursion.

```
Hubs confer prestige to authorities:
    $a(v) \leftarrow \sum_{(u,v)\in E} h(u)$ for all $v$
Total prestige is normalized:
    $\Sigma_a \leftarrow \sum_v a(v)$
    $a(v) \leftarrow a(v)/\Sigma_a$ for all $v$
Authorities reflect prestige to hubs [28])
    $h(u) \leftarrow \sum_{(u,v)\in E} a(v)$ for all $u$
Total reflected prestige is normalized:
    $\Sigma_h \leftarrow \sum_u h(u)$
    $h(u) \leftarrow h(u)/\Sigma_h$ for all $u$
```

If $\mathbf{E}$ is the adjacency matrix for E, h and a converge to the dominant eigenvectors of $\mathbf{E}^T\mathbf{E}$ and $\mathbf{E}\mathbf{E}^T$. Pages having large a values are highly popular *authorities*, and pages having large h are good resource lists or *hubs.*

2.2.2 Enhancements for resource discovery

For the purpose of topic-driven discovery, some important enhancements are needed. In the above procedure, each edge is implicitly assumed to have the same importance. Some limitations of this assumption have been described in later work [7, 4], which have assigned various heuristic weights, based on the keyword query, to the edges to improve precision. In our setting there is no query, but there are topics induced by examples, and we wish to model the strength of hyperlinks using the relevance judgment.

To appreciate the model that we will propose, observe that w.r.t. almost any topic, relevant pages refer to irrelevant pages and vice versa with appreciable frequency, owing to the diversity of authorship. Pages of all topics point to Netscape and Free Speech Online. Conversely, bookmark files that are great resources about sports cars may also have links to photography sites.

We will specialize the forward and backward adjacency matrices $\mathbf{E}$ and $\mathbf{E}^T$ into two differently weighted matrices $\mathbf{E}_F$ and $\mathbf{E}_B$. We propose that the weight $\mathbf{E}_F[u, v]$ of edge (u, v) be the probability that u linked to v because v was relevant to the topic, i.e., `relevance`(v). This has the effect of preventing leakage of endorsement or prestige from relevant hubs to irrelevant authorities. Similarly, we propose that $\mathbf{E}_B[u, v]$ be set to `relevance`(u), to prevent a relevant authority from transferring prestige to an irrelevant hub. Another effect that has to be corrected for is *inflation* of endorsement. This is done similar to Bharat et al [4].

2.2.3 I/O efficient implementation

Two tables `HUBS` and `AUTH` are used to perform distillation. They have the same schema: a 64-bit hashed `oid` key for the URL (u and v in the formulae before) and a floating point field `score` representing h and a as per context. The third table involved in distillation is the `LINK` table, which has six attributes.

`oid_src`: ID corresponding to source URL, u in previous formulae.

`sid_src`: The server (represented by IP address) that served u. This is not always fool-proof, because of DNS-based load-balancing, multi-homed hosts, etc., but these aberrations were tolerable.

`oid_dst`: ID of target URL, or v in formulae.

`sid_dst`: Server of target URL.

`wgt_fwd`: Forward iteration edge weight, or $\mathbf{E}_F[u, v]$ in formulae (see §2.2).

`wgt_rev`: Reverse or backward iteration edge weight, or $\mathbf{E}_B[u, v]$ in formulae (see §2.2).

In past work on distillation, the graphs had few hundred nodes and iterations were done within main memory. An array of links would be traversed, reading and updating the endpoints using node hashes. We estimate that a graph would $|V|$ nodes would need about $336|V|$ bytes of RAM. Large graphs would not fit in memory, which is shared with the classifier and crawler. Furthermore, to keep the crawl persistent and influence the crawler's decisions, we would have to write out the new scores to disk anyway.

A cleaner approach is to write the distillation as another database application, so that depending on the graph size, the database would automatically pick an I/O-efficient execution plan. This also enables running distillation concurrently with the crawler and classifier. Furthermore, distillation can then be triggered by substantial changes in the crawl graph. The code for one iteration of the distiller is shown in Figure 4. Notice how it is asymmetric w.r.t. update of `HUBS` and `AUTH`.

3 Experiments

Our prototype crawler is a C++ ODBC/CLI application that was run on a dual-processor 333 MHz Pentium-II PC with 256 MB of RAM and SCSI disk. IBM Universal Database v5 and v5.2 were used. The administration program is a JDBC-based Java applet. Our test machines are connected through a half-duplex 10 MB/s Ethernet through the router to a SOCKS firewall machine. The firewall is connected to the ISP using full-duplex 10 MB/s SMDS over DS3. The ISP connects us to a 622 MB/s OC12 ATM backbone (UUNET High Performance Network). Comparisons between standalone mining and mining written on top of the database were done on a 266 MHz Pentium-II PC with 128 MB or RAM.

3.1 Using a DBMS

We started building our prototype as a C++ application using the file system to maintain crawl state [10]. As we made progress, many services provided by a relational engine became essential. The crawler is multithreaded; these threads concurrently access the unexplored crawl frontier stored on disk. Few pages on the Web are formally checked for well-formedness, hence all crawlers crash [5]. Keeping all crawl tables and indices consistent by hand amounted to reinventing the wheel.

Robust data storage was the initial reason for using a DBMS, but we quickly realized that we could exploit other features. It became trivial to write ad-hoc SQL queries to monitor the crawler and diagnose problems such as stagnation. In most cases, the queries we asked were not planned ahead of time. Multiple index orders could be implemented on the crawl frontier, and the policy could even be changed dynamically. Such experiments would cost major coding effort in the case of a standalone application.

Gradually we started using the DBMS in more advanced ways. We rewrote the classifier and distiller to maximally exploit the I/O efficiency of sort-merge joins. This increased our discovery rate by almost an order of magnitude. We also used triggers to recompute relevance and centrality scores when the neighborhood of a page changed significantly owing to continued crawling.

3.2 Controlling the crawler

Now we will describe how the scores determined by the classifier and distiller are combined with other per-URL and per-server statistics to guide the crawler. To make the discussion concrete, we give a specific design, but it is important to note the flexibility of the architecture to supporting other policies and designs as well.

There are three numbers associated with each page u: the relevance $R(u)$, the hub score $h(u)$ and the

UpdateHubs

```
delete from HUBS;
insert into HUBS(oid, score)
   (select oid_src, sum(score * wgt_rev)
   from AUTH, LINK
   where sid_src <> sid_dst /* avoid nepotism */
   and oid = oid_dst
   group by oid_src);
update HUBS set (score) = score /
   (select sum(score) from HUBS)
```

UpdateAuth(ρ)

```
delete from AUTH;
insert into AUTH(oid, score)
   (select oid_dst, sum(score * wgt_fwd)
   from HUBS, LINK, CRAWL
   where sid_src <> sid_dst
   and HUBS.oid = oid_src and oid_dst = CRAWL.oid
   and relevance > ρ /* filter */
   group by oid_dst);
update AUTH set (score) = score /
   (select sum(score) from AUTH )
```

Figure 4: SQL code for distillation to find relevant authorities and hubs. Their scores are used to modify crawl (re)visit priorities.

authority score $a(u)$. $R(u)$ is in the `CRAWL` table, and $h(u)$ and $a(u)$ are the `score` fields in the `HUBS` and `AUTH` tables. Apart from these numbers, we need a few other numbers in the `CRAWL` table to control the crawl. The first is `numtries`, which records the number of times the crawler attempted to fetch the URL u. The second is `serverload`, which is a crude and lazily updated estimate of the number of distinct URL's fetched from the same server as u. Its purpose is to prevent the crawler going depth-first into one or a few sites.

In aggressive discovery mode, the highest priority is seeking out new resources. New work is checked out from the `CRAWL` table in the order

```
(numtries ascending, relevance descending,
    serverload ascending).
```

Occasionally, `HUBS.score` is used to trigger the raising of `relevance` of unvisited pages cited by some of the top hubs. We have not had much experience in crawl maintenance, but lexical orderings such as

```
(lastvisited ascending, HUBS.score descending), or
(numtries descending, AUTH.score descending,
    relevance descending)
```

behaved reasonably, provided timeouts and dead links (very high `numtries`) were picked off separately. In production runs, additional criteria can be useful, for instance, an estimate of the average interval between updates to a page that has already been visited. But notice that the code change would be minimal.

A crawler can use various devices to extend its frontier. Typically, it scans each fetched page for outgoing hyperlink URL's. However, other strategies are also known. E.g., if the URL is of the form `http://`*`host`*`/`*`path`*, the crawler may truncate components of *`path`* and try to fetch these URL's. If links could be traversed *backward*, e.g. using metadata at the server [9], the crawler may also fetch pages that point to the page being 'expanded.'

3.3 Evaluation setup

We picked about twenty topics that could be represented by one or few nodes in a master category list derived from Yahoo!, such as gardening, mutual funds, cycling, HIV, etc. On most of these topics, the main performance indicators were comparable, so we present a representative sample of results. We were limited only by experimentation time: we did not want to overload the network and disrupt our firewall. A full-scale crawler never operates through a firewall. Although we had access to machines outside the firewall, we decided to demonstrate the viability of our system by running it inside the firewall and consuming negligible network resources. We ran the crawler with relatively few threads compared to what it can handle. In our opinion, it was more important to study the behavior of the system for many different topics, than study extremely large crawls, although a few crawls were left running for days.

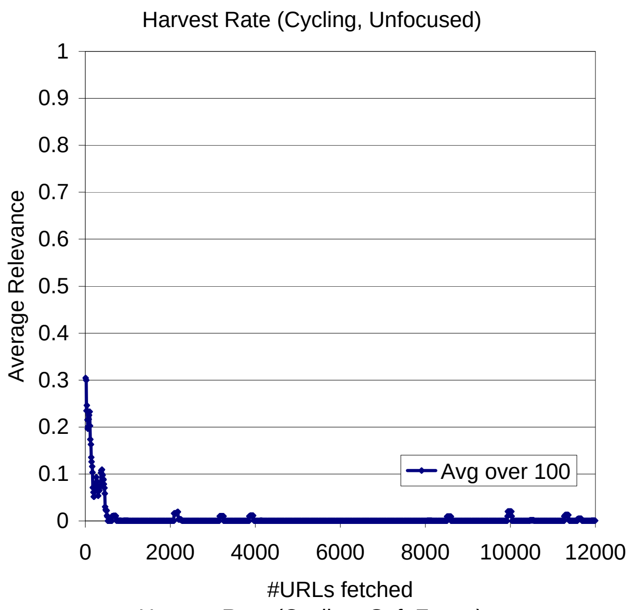
3.4 Harvest rate or precision

By far the most important indicator of the success of our system is the *harvest rate*, or the average fraction of crawled pages that are relevant. We want the crawler to spend most of its time acquiring useful pages, not eliminating irrelevant pages.

Human judgment, although subjective and even erroneous, would be best for measuring relevance. Clearly, even for an experimental crawler that acquires only ten thousand pages per hour, this is impossible. Therefore we use our classifier to estimate the relevance of the crawl graph. This methodology may appear flawed, but is actually not flawed. It is to be noted carefully that we are not, for instance, training and testing the classifier on the same set of documents, or checking the classifier's earlier evaluation of a document using the classifier itself.

Just as human judgment is prone to variation and error [25], we have a statistical program that makes mistakes. Based on such imperfect recommendation, we choose to or not to expand pages. Later, when a page that was chosen is visited, we evaluate its relevance, and thus the value of that decision. Thus we are evaluating not the classifier but the validity and viability of the architecture.

Representative crawls on bicycling starting from the result of topic distillation with keyword search `cycl* bicycl* bike` are studied in Figure 5(a) (standard



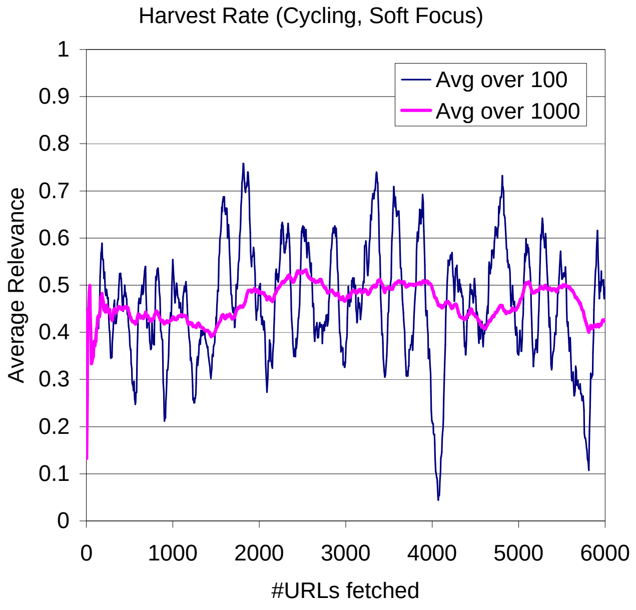
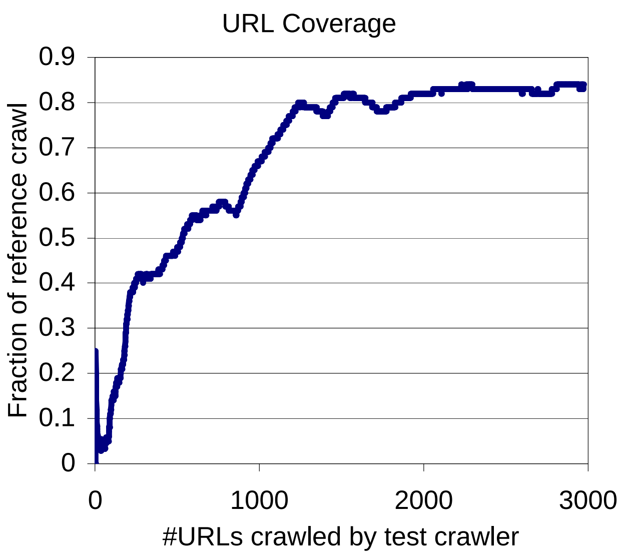


Figure 5: Our system acquires relevant pages at a high harvest rate, whereas a standard crawler starting at the same set of URL's quickly loses its way. The relevance of pages acquired by our system is typically three orders of magnitude higher than a standard crawler.

crawler) and (b) (our system). More extensive experiments with other topics are reported elsewhere [10]. The x-axis shows the number of pages acquired (as a representative of real time). The y-axis shows a moving average of $R(p)$, where p represents pages collected within the window. It is immediately evident that relevant resource discovery does not happen by accident; it has to be done very deliberately. The standard crawler starts out from the same set of dozens of highly relevant links as our crawler, but is completely lost within the next hundred page fetches: the relevance



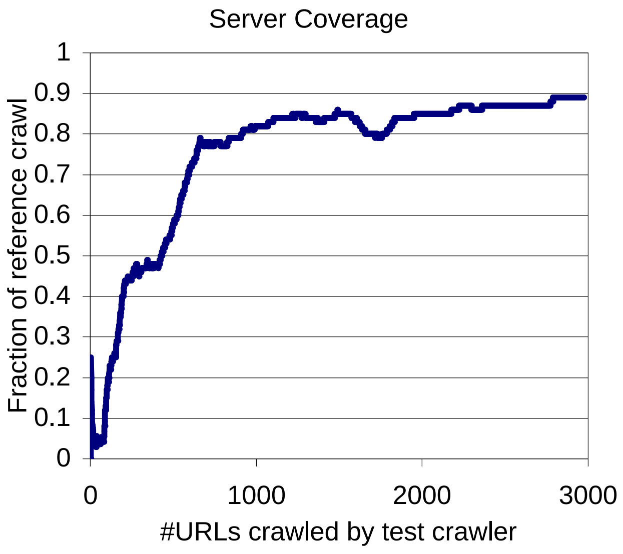


Figure 6: Coverage experiment. A reference crawl is prepared by running our crawler from one start set. Then a disjoint second start set is picked and a test crawl started to see how fast it visits the relevant URL's (a) and web sites (b) visited by the reference crawler.

goes quickly toward zero. In contrast, it is heartening to see that our crawler keeps up a healthy pace of acquiring relevant pages. On an average, every second page is relevant.

3.5 Estimating recall or coverage

For a closed, self-contained data set, precise measurements of recall or coverage can be made. For the web, recall is essentially impossible to assess. However we must produce some reasonable evidence of robust coverage. So we take recourse to the following measurement, similar to Cho et al [13]. We first build a *reference crawl* by selecting a random set S_1

of start URLs from a set of sources, e.g., Yahoo!, Infoseek, and Excite. We run our crawler starting from S_1 for some fixed time (one hour). Then we collect another random set S_2 of start sites from Alta Vista, making sure that $S_1 \cap S_2 = \emptyset$, i.e. the start sets are disjoint. Then we start a separate crawl from S_2, monitoring along time the fraction of the relevant[16] URLs in the reference crawl that are visited by the second test crawl. This ought to give a reasonable feel for how robust the system is. We used a relevance threshold of $\log R(u) > -1$ to include a page u, but the conclusions are not sensitive to this choice. The results are shown in Figure 6(a). It is encouraging to see that within an hour of crawling, the test crawler collects up to 83% of the relevant URLs collected by the reference crawler. It is also important to measure the rate at which *web servers* visited by the reference crawl are visited for the first time by the test crawl; this is shown in Figure 6(b). Within an hour, this number reaches 90%.

3.6 Evidence of large-radius exploration

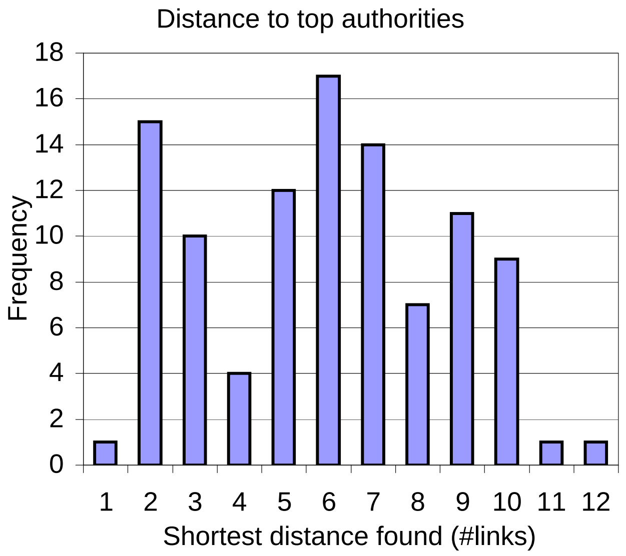
After one hour of crawling, we collected the top hubs and authorities from the crawl. We list the hubs for cycling in Figure 7 and strongly encourage the reader to follow these links (verified to be accessible on February 22, 1999). How do these compare with traditional topic distillation? We need to provide evidence that we did not unduly help our system by starting it at or near some of the best sites above. To do this, we will plot histograms of the shortest distance (number of links) of the top 100 authorities from the start set. If most of the best authorities are very close to the start set, we cannot claim significant value in the goal-driven exploration. Fortunately, the plots in Figure 7 suggest that this is not the case: excellent resources were found as far as 12–15 links from the start set. Often, there are millions of pages within such distances of any web page. Therefore, our system was performing nontrivial on-line filtering, which was crucial to identifying these resources by crawling only about 6000 pages.

Thus, distillation applied to the goal-directed crawl performs qualitatively better than distillation applied to the result of keyword search, which reconfirms the value of our approach. However, note that superior distillation is just one application of resource discovery. We envisage that a standard search over the corpus, or unsupervised clustering, are likely to be much more satisfying in the scope of the focused corpus. We will explore these in future work.

3.7 Crawl monitoring and tweaking

The ease with which we wrote ad-hoc utilities to monitor the crawler demonstrated the value of using a relational database. We created an applet with a JDBC connection to `CRAWL` to plot Figure 5 continuously. The query was simply

[16]It does not matter if the irrelevant pages are different.



```
http://www.truesport.com/Bike/links.htm
http://reality.sgi.com/billh_hampton/jrvs/links.html
http://www.acs.ucalgary.ca/~bentley/mark_links.html
http://www.cascade.org/links.html
http://www.bikeride.com/links/road_racing.asp
http://www.htcomp.net/gladu/'drome/
http://www.tbra.org/links.shtml
http://www.islandnet.com/~ngs/SVCyclingLinks.html
http://www.novit.no/dahls/Cycling/hotlist.html
http://members.aol.com/velodromes/MajorTaylor/links.htm
http://www.nashville.com/~mbc/mbc.html
http://www.bikindex.com/bi/races.asp
http://www.louisvillebicycleclub.org/links.htm
http://world.std.com/~nebikclb/misc/netresources.html
http://crisny.org/not-for-profit/cbrc/links.htm
http://members.aol.com/velodromes/index.htm
```

Figure 7: Histogram of shortest number of links to the top 100 authorities on cycling and a few top hubs, found after one hour (6000 page fetches). The reader is encouraged to follow these links.

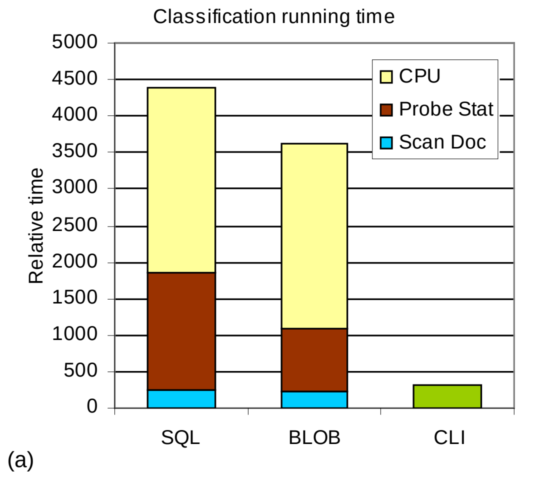
```
select minute(lastvisited), avg(exp(relevance))
from CRAWL
where lastvisited + 1 hour > current timestamp
group by minute(lastvisited)
order by minute(lastvisited).
```

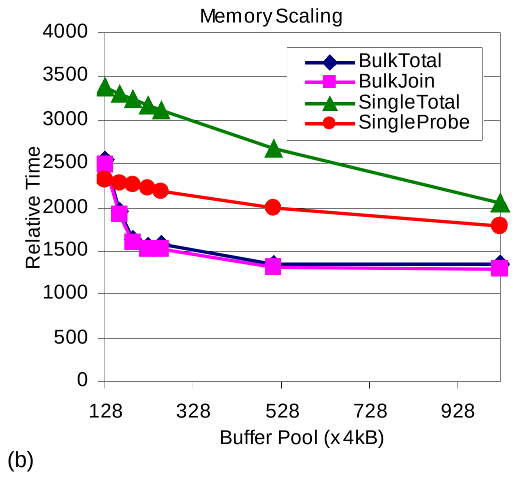
Only one crawl dropped in relevance (mutual funds). To diagnose why, we asked:

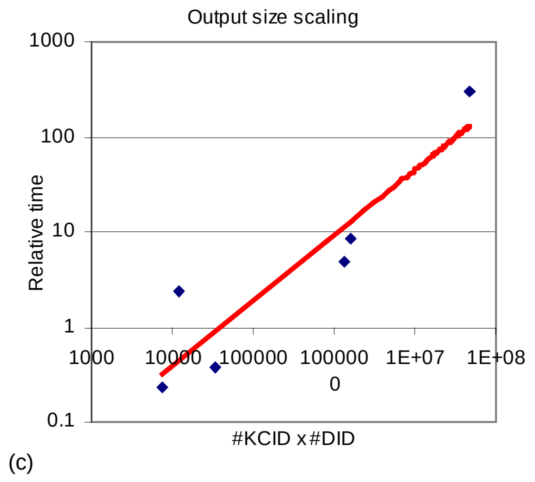
```
with CENSUS(kcid, cnt) as
  (select kcid, count(oid) from CRAWL group by kcid)
select kcid, cnt, name from CENSUS, TAXONOMY
  where CENSUS.kcid = TAXONOMY.kcid order by cnt
```

This query immediately revealed that the neighborhood of most pages on mutual funds contained pages on *investment* in general, which was an ancestor of mutual funds. One update statement marking the ancestor `good` fixed this stagnation problem. Finally, we will give an example of how the distillation program can help the crawler modify its priority to get good pages it was otherwise neglecting. Suppose ψ is the 90th percentile of hub scores. To ask about possibly missed neighbors of great hubs, we write

```
select url, relevance from CRAWL where oid in
  (select oid_dst from LINK
    where oid_src in
```







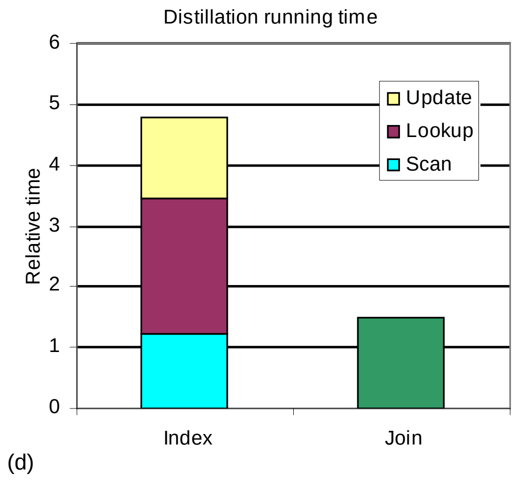


Figure 8: Performance of I/O conscious algorithms against simpler equivalents. (a), (b) and (c) show classification and (d) shows distillation results. (a) shows running time of **SingleProbe** with SQL (left) and BLOB (middle) broken down into CPU time, **PROBE** time, and document scanning time; and **BulkProbe** (marked CLI). (b) shows how **SingleProbe** (BLOB) and **BulkProbe** scale with memory (a smaller dataset was used to collect many data points for **SingleProbe** which is slow). **SingleProbe** shows poor utilization of additional buffer pool whereas **BulkProbe** has better locality. (c) shows that the running time of **BulkProbe** is essentially proportional to output size. (d) compares the running time of naive distillation using sequential link table scan against a better join-based implementation. Time is broken down into scanning of `LINK`, index lookups for `HUBS` and `AUTH` tables, and update of authority and hub scores.

```
        (select oid from HUBS
        where score > ψ)
      and sid_src <> sid_dst)
    and numtries = 0
```

3.8 I/O performance

Figure 8(a) shows the performance of three variants of the classifier. The second bar (BLOB) uses the `BLOB` tables, the first (SQL) and third (CLI) bars don't. The first and second measure **SingleProbe**; the third measures **BulkProbe**. Relative time per document is charted, broken down when appropriate into time for reading `DOCUMENT`, time for computation, and time for probing the statistics. Over an order of magnitude reduction in overall running time is seen using the bulk formulation. Figure 8(b) plots relative running time per document as the buffer pool size is varied, for **SingleProbe** and **BulkProbe**. Because there is little locality, **SingleProbe** shows continual reduction in running time as buffer pool is increased. For much smaller buffer pool size, the running time of **BulkProbe** steeply drops and stabilizes, showing the superior memory usage of the bulk approach. Figure 8(c) shows, for various c_0's and sets of documents $\{d\}$, a scatter of running times against the product $|\{c_i\}|\,|\{d\}|$ (the output size). We see that the bulk algorithm is roughly linear in output size. Figure 8(d) shows the performance of two variants of the distiller: one derived from sequential edge-list walking as in earlier main-memory implementations, the other expressed as a join as in Figure 4. For the former, time is broken down into edge scan, end-vertex index lookup, and score updates. The join approach is a factor of three faster.

4 Conclusion

We have demonstrated that goal-directed web resource discovery is a powerful means to structure web content so that questions combining structured linkage and meta-data with unstructured topic information can be answered efficiently. We have architected such as resource discovery system around a relational database, using it not only as a robust data repository but also as an I/O-efficient hypertext mining engine. It would be interesting to further automate the administration of the system.

Acknowledgements: We are grateful to Global Web Solutions, IBM Atlanta, for partially funding this project. We thank Tom Mitchell, Dan Oblinger and Steve Gates for helpful discussions, Myron Flickner for generously contributing disks and computers, David Gibson for contributing code to the Java user interface, and Sunita Sarawagi, Amit Somani and Kiran Mehta for advice with DB2/UDB.

References

[1] C. Apte, F. Damerau, and S. M. Weiss. Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems*, 1994. IBM Research Report RC18879.

[2] I. Ben-Shaul, M. Herscovici, M. Jacovi, Y. S. Maarek, D. Pelleg, M. Shtalheim, V. Soroka, and S. Ur. Adding support for dynamic and focused search with Fetuccino. In *8th World Wide Web Conference*. Toronto, May 1999.

[3] K. Bharat and A. Broder. A technique for measuring the relative size and overlap of public web search engines. In *Proceedings of the 7th World-Wide Web Conference (WWW7)*, 1998. Online at http://www7.scu.edu.au/programme/fullpapers/1937/com1937.htm; also see an update at http://www.research.digital.com/SRC/whatsnew/sem.html.

[4] K. Bharat and M. Henzinger. Improved algorithms for topic distillation in a hyperlinked environment. In *Proceedings of the 21st International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 469–477, 1998. Online at http://www.research.digital.com/SRC/personal/monika/papers/sigir98.ps.gz.

[5] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the 7th World-Wide Web Conference (WWW7)*, 1998. Online at http://decweb.ethz.ch/WWW7/1921/com1921.htm.

[6] S. Chakrabarti, B. Dom, R. Agrawal, and P. Raghavan. Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. *VLDB Journal*, Aug. 1998. Invited paper.

[7] S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, P. Raghavan, and S. Rajagopalan. Automatic resource compilation by analyzing hyperlink structure and associated text. In *Proceedings of the 7th World-wide web conference (WWW7)*, 1998. Online at http://www7.scu.edu.au/programme/fullpapers/1898/com1898.html and at http://www.almaden.ibm.com/cs/people/pragh/www98/438.html.

[8] S. Chakrabarti, B. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. In *SIGMOD*. ACM, 1998. Online at http://www.cs.berkeley.edu/~soumen/sigmod98.ps.

[9] S. Chakrabarti, D. Gibson, and K. McCurley. Surfing the web backwards. In *8th World Wide Web Conference*, Toronto, Canada, May 1999.

[10] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: A new approach to topic-specific resource discovery. In *8th World Wide Web Conference*, Toronto, May 1999.

[11] D. Chamberlin. *A complete guide to DB2 universal database.* Morgan-Kaufmann, 1998.

[12] C. Chekuri, M. Goldwasser, P. Raghavan, and E. Upfal. Web search using automatic classification. In *Sixth World Wide Web Conference*, San Jose, CA, 1996.

[13] J. Cho, H. Garcia-Molina, and L. Page. Efficient crawling through URL ordering. In *7th World Wide Web Conference*, Brisbane, Australia, Apr. 1998. Online at http://www7.scu.edu.au/programme/fullpapers/1919/com1919.htm.

[14] W. W. Cohen. Fast effective rule induction. In *Twelfth International Conference on Machine Learning*, Lake Tahoe, CA, 1995. Online at http://www.research.att.com/~wcohen/postscript/ml-95-ripper.ps and http://www.research.att.com/~wcohen/ripperd.html.

[15] J. Dean and M. R. Henzinger. Finding related pages in the world wide web. In *8th World Wide Web Conference*, Toronto, May 1999.

[16] P. DeBra and R. Post. Information retrieval in the world-wide web: Making client-based searching feasible. In *Proceedings of the First International World Wide Web Conference*, Geneva, Switzerland, 1994.

[17] S. Dumais, J. Platt, D. Heckerman, and M. Sahami. Inductive learning algorithms and representations for text categorization. In *7th Conference on Information and Knowledge Management*, 1998. Online at http://www.research.microsoft.com/~jplatt/cikm98.pdf.

[18] R. Goldman, N. Shivakumar, S. Venkatasubramanian, and H. Garcia-Molina. Proximity search in databases. In *VLDB*, volume 24, pages 26–37, New York, Sept. 1998. Online at http://www-db.stanford.edu/pub/papers/proximity-vldb98.ps.

[19] J. Hammer, H. Garcia-Molina, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. Information translation, mediation, and mosaic-based browsing in the TSIMMIS system. In *SIGMOD Exhibit*, page 483, San Jose, CA, June 1995. Online at ftp://www-db.stanford.edu/pub/papers/mobie-demo-proposal.ps.

[20] T. Joachims, D. Freitag, and T. Mitchell. WebWatcher: A tour guide for the web. In *IJCAI*, Aug. 1997. Online at http://www.cs.cmu.edu/~webwatcher/ijcai97.ps.

[21] L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, Mar. 1953.

[22] T. Kistler and H. Marais. WebL—a programming language for the web. In *7th World Wide Web Conference*, Brisbane, Australia, 1998. Online at http://www.research.digital.com/SRC/personal/Johannes_Marais/pub/www7/paper.html and http://www.research.digital.com/SRC/WebL.

[23] J. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, 1998. Also appears as IBM Research Report RJ 10076(91892), and online at http://www.cs.cornell.edu/home/kleinber/auth.ps.

[24] D. Konopnicki and O. Shmueli. WWW information gathering: The W3QL query language and the W3QS system. *TODS*, 1998. Online at http://www.cs.technion.ac.il/~konop/todsonline.ps.gz.

[25] S. Macskassy, A. Banerjee, B. Davidson, and H. Hirsh. Human performance on clustering web pages: A performance study. In *Knowledge Discovery and Data Mining*, volume 4, pages 264–268, 1998.

[26] A. Mendelzon and T. Milo. Formal models of the web. In *PODS*, Tucson, AZ, June 1997. Online at ftp://ftp.db.toronto.edu/pub/papers/pods97MM.ps.

[27] A. Mendelzon and T. Milo. Formal models of the web. In *PODS*, Tucson, Arizona, June 1997. ACM. Online at ftp://ftp.db.toronto.edu/pub/papers/pods97MM.ps.

[28] M. S. Mizruchi, P. Mariolis, M. Schwartz, and B. Mintz. Techniques for disaggregating centrality scores in social networks. In N. B. Tuma, editor, *Sociological Methodology*, pages 26–48. Jossey-Bass, San Francisco, 1986.

[29] W. Niblack, X. Zhu, J. Hafner, T. Breuel, D. Ponceleon, D. Petkovic, M. Flickner, E. Upfal, S. Nin, , S. Sull, B. Dom, B. Yeo, S. Srinivasan, D. Zivkovic, and M. Penner. Updates to the QBIC system. In *Storage and Retrieval for Image and Video Databases VI*, volume 3312 of *Proceedings of SPIE*, Jan. 1998.

[30] M. Pazzani, L. Nguyen, and S. Mantik. Learning from hotlists and coldlists: Towards a www information filtering and seeking agent. In *Seventh International Conference on Tools with Artificial Intelligence*, 1995. Online at http://www.ics.uci.edu/~pazzani/Publications/Coldlist.pdf.

[31] J. Savoy. An extended vector processing scheme for searching information in hypertext systems. *Information Processing and Management*, 32(2):155–170, Mar. 1996.

[32] L. Terveen and W. Hill. Finding and visualizing intersite clan graphs. In *Computer Human Interaction (CHI)*, pages 448–455, Los Angeles, CA, Apr. 1998. ACM SIGCHI. Online at http://www.research.att.com/~terveen/chi98.htm and http://www.acm.org/pubs/articles/proceedings/chi/274644/p448-terveen/p448-terveen.pdf.

[33] S. Wasserman and K. Faust. *Social Network Analysis.* Cambridge University Press, 1994.