Index-based Persistent Document Identifiers[*][†]

Diomidis Spinellis
Department Management Science and Technology
Athens University of Economics and Business[‡]

December 4, 2004

Abstract

The infrastructure of a typical search engine can be used to calculate and resolve persistent document identifiers: a string that can uniquely identify and locate a document on the Internet without reference to its original location (URL). Bookmarking a document using such an identifier allows its retrieval even if the document's URL, and, in many cases, its contents change. Web client applications can offer facilities for users to bookmark a page by reference to a search engine and the persistent identifier instead of the original URL. The identifiers are calculated using a global Internet term index; a document's unique identifier consists of a word or word combination that occurs uniquely in the specific document. We use a genetic algorithm to locate a minimal unique document identifier: the shortest word or word combination that will locate the document. We tested our approach by implementing tools for indexing a document collection, calculating the persistent identifiers, performing queries, and distributing the computation and storage load among many computers.

"Users should beware that there is no general guarantee that a URL which at one time points to a given object continues to do so, and does not even at some later time point to a different object due to the movement of objects on servers."

— T. Berners-Lee et al. *Uniform Resource Locators (*URL*).* RFC 1738.

1 Introduction

Internet resources are typically specified using the string representation of "Uniform Resource Locators" (URLs). URLs are a subset of the Uniform Resource Identifiers

[*] *Information Retrieval,* 8(1):5–24, January 2005.

[‡] Address: Patision 76, GR-104 34 Athina, Greece. email: dds@aueb.gr

(URIs) that provide an abstract identification of a resource location (Berners-Lee *et al.*, 1994). URLs are often used to identify resources in hypertext links, printed media such as business cards, billboards, and publications, and in user-maintained collections such as bookmarks and visited site history files.

The dynamic nature of the web—Chankhunthod *et al.* (1996) report the average lifetime of an HTML text object to be 75 days—results in URLs that quickly decay and become inaccessible (Pitkow, 1999; Ashman, 2000). According to our earlier work (2003) around 27% of the URLs referenced in *IEEE Computer* and the *Communications of the ACM* articles from 1995–2000 were no longer accessible at the end of the period. In addition, after four years almost 50% of the referenced URLs are inaccessible. Lawrence *et al.* (2001) have identified similar trends for published URLs. A number of solutions have been proposed for handling this *link integrity* or *referential integrity* problem. The solution classes that have been identified (Ashman, 2000) include prohibiting change, maintaining document versions, regularly updating all links, aliasing a link's end points, posting notifications of changes to document locations, implementing forwarding mechanisms, automatically detecting and correcting broken links, and creating all links dynamically. For links to published papers citation linking (Hitchcock *et al.*, 1999), provided as a service by publishers or public-service efforts (Lawrence *et al.*, 1999), may lead to publication formats that actively support hypertext links across time.

Although URNs, PURLs and the Handle mechanism may offer a long-term solution, they have up to now not been universally adopted. Thus, individual user bookmarks and publicly distributed URLs quickly become obsolete as documents change names, directories, or are hosted on different places. Although a search engine (Lawrence & Giles, 1999; Takeda, 2000) can be used to relocate a document after a web server's "404—document not found" response, this is a tedious and error-prone procedure. In this paper we describe a way to automate the searching task by providing a more persistent alternative to a URL. Such an alternative can be used both to provide page bookmarks that are relatively immune to URL changes and as a centralized, alternative method for creating URNs without the active cooperation of the content creators.

Our scheme involves having a search engine calculate for every URL an augmented, persistent version. The augmented URL, containing the original URL and words that uniquely identify the document, will have a high probability to locate the original document even if its contents have changed location. Users who save the persistent URL to bookmark a page can later transparently retrieve the document through a search engine's infrastructure. If the default document retrieval mechanism fails, the search engine will resolve the URL by searching for documents containing the words embedded in the URL. In addition, web sites can use persistent URLs to point to pages outside their administrative domain with a lower probability that these links will become unavailable when the respective page contents change location. As an example, given the URL http://moving.org/target whose contents could be uniquely identified by the words *gloxinia* and *obelisk* the corresponding persistent URL would be of the form: http://-resolve.com/find?orig=http://moving.org/target&w=gloxinia+obelisk (to make the example clearer we have not URL-encoded the original URL). When the user tries to access the above URL the search engine infrastructure at resolve.com will first try to retrieve the document at http://moving.org/target. If that fails, it will search its (up to date) index for a document containing the words *gloxinia* and *obelisk*. If one of the two above actions succeeds the user will be redirected to the document's original or revised location, otherwise a “404 Not found” error will be returned.

The remainder of this paper outlines the current process of document retrieval and the associated errors (Section 2), describes our algorithm for calculating unique document discriminants (Section 3), and sketches a prototype implementation of the concept (Section 4). In Section 5 we discuss the method’s performance in terms of retrieval accuracy, time and space requirements, and scalability. The paper concludes with a presentation of possible extensions and applications of our technique.

2 Web Document Retrieval

In general, URLs consist of a *scheme* (e.g. http, ftp, mailto) followed by a colon and a scheme-specific part. The syntax of the scheme-specific part can vary according to the scheme. However, URL schemes that involve direct use of an IP-based protocol to an Internet host use the following common syntax:

```
//<user>:<password>@<host>:<port>/<url-path>
```

The double slash indicates that the scheme data complies with the Internet scheme syntax. The host is specified using the fully qualified domain name of a network host or its IP address. The default port for the HTTP scheme is 80 and is usually omitted.

For a web page of a given URL to appear on a browser’s screen a number of different technologies and protocols must work in concert. In addition, the changing realities of the web and the Internet have vastly complicated the simple end-to-end request-reply protocol that used to form the basis of the early HTTP transactions. Any failure along the complicated chain of actions needed to retrieve a web page will lead to a failed URL reference.

The path appearing in a URL will nowadays not necessarily match with a corresponding local file on the server. Web servers provide a number of mechanisms for managing namespaces. Some of them are: the creation of a separate namespace for every local user, the definition of protection domains and access mechanisms, the support of aliases to map namespaces to local directories, and the dynamic creation of content using technologies such the common gateway interface (CGI), servlets, and server-modified pages (ASP, JSP, PSP). In addition, a feature of the HTTP protocol called *content negotiation* allows a server to provide different pages based on technical or cultural characteristics of the incoming request (e.g. bandwidth, display technology, languages the user can understand).

The HTTP protocol defines 24 different errors that can occur within an HTTP exchange. In addition, some errors can occur before the client and server get a chance to communicate. In practice, while verifying thousands of published URLs we encountered the following errors:

400 Bad request The syntax used for the request could not be understood by the server. This may signify a badly formed URL often coupled with a browser bug.

401 Unauthorized The request requires user authentication. Such an error can result when citations are given to URLs that exist within a domain of services that require registration, or when such services move from a free access to a registration-based model.

403 Forbidden The server is refusing to fulfill the given request, in this case however proper authorization can not be used to retrieve the page. It is conceivable that URLs that are not part of the public Internet end up as citations when the authors fail to realize that they have special privileges to access certain repositories that do not apply to the global Internet population. As an example, our organization has transparent access to a collection of on-line journals with authentication based on the client IP address. URLs to this collection provided by unsuspecting users will typically generate a 403 error.

404 Not Found This infamous and quite common response signifies that the server has not found anything matching the Request-URI. This error is typically generated when web site maintainers change file names that are part of the given URL path or entirely remove the referenced material. Note that this protocol error can be followed by customized content—typically HTML text that informs the user of the problem and provides alternative navigation options.

500 Internal Server Error The server encountered an unexpected condition which prevented it from fulfilling the request. This error can occur when a server is wrongly configured, or, more commonly, if a program or database that is used to serve dynamic content fails.

503 Service Unavailable The server is currently unable to handle the request due to temporary overloading or maintenance of the server. Errors of this type sometimes appear on a misconfigured server, or servers overwhelmed by traffic.

504 Gateway Time-out The server, acting as a proxy or gateway did not receive a timely response from the upstream server specified by the URI (e.g. HTTP, FTP) or some other auxiliary server (e.g. domain name server—DNS) it needed to access in attempting to complete the request. When HTTP requests are transparently intercepted by a proxy caching server, network connectivity problems are likely to appear as 504 errors.

901 Host lookup failure The host name could not be mapped to an IP address. This error (which is not part of the HTTP protocol) signifies a problem in retrieving the IP address of the server using the DNS services. Likely causes include changes of host names, and DNS server failures or connectivity problems.

Not all of the above problems can be solved by the provision of persistent URLs. A persistent URL will help in cases where the original document has changed its name (including the path to its name) resulting in a “404 Not found” error, and in cases where the domain hosting the URL is renamed resulting in a “901 Host lookup failure” error. Changes in a document’s access authorization are not a technical but a legal problem, while the 500-class server errors are more appropriately handled by a robust network infrastructure and mechanisms such as proxies, mirrors, archives, and content delivery networks. In addition, persistent URLs can not deal with documents that are deleted or modified, unless the corresponding URLs are designed to work in concert with an appropriate archive repository.

3 Unique Document Discriminants

Our method for creating persistent URLs involves calculating a minimal set of identifying words that can be used to uniquely select a given document in a search engine query. Consider three documents and their respective word contents $1 : ABCF$, $2 : ABDF$, and $3 : AKCDF$. The minimal unique identifiers (discriminants) for these documents are for document $1 : A \wedge B \wedge C$, for document $2 : D \wedge B$, and for document $3 : K$. In calculating the minimal discriminants we also take into account the length of each word to minimize the length of the respective search engine query. Given such a discriminant a search engine query with that discriminant, will result in a single matching element: the identified document, irrespective of the document's location (URL). As an example our application calculated that the words *sedam protectable* currently uniquely identify the web page http://research.unc.edu/otd/inventors/overview.html. Thus a search engine query for the above terms will result in a single result: the corresponding document. We theorize that for large collections and a regularly updated search engine index, the discriminants will continue to uniquely and correctly identify the document, even as new documents are added to the collection. The form submission mechanism of many search engines allows one to form a URL that will automatically perform the above search and return the corresponding result; as an example the Google search engine URL for the page we described would be http://www.google.-com/search?&q=sedam+protectable. Furthermore, a simple addition of an appropriate redirection header to the query results would even allow the entire operation to be transparently performed.

The computing infrastructure of a large search engine is ideally placed to efficiently perform both the calculation of the persistent URLs and their resolution. These two can then be provided as an extra service to the engine's users. In addition, the search engine will draw additional traffic (and potentially advertising revenues) each time a user accesses a bookmarked persistent URL or follows such a URL on the web.

The persistent URLs will most likely be less easy to remember than the URLs they are derived from. However, these URLs will be primarily stored as bookmarks and hyperlinks and automatically processed, rather than memorized and communicated by humans. For this reason, it is not necessary to use natural terms for searching; any terms that uniquely identify the document are suitable for this purpose.

The idea of locating documents by words those documents contain is not novel. Phelps and Wilensky (2000) proposed the construction of *robust hyperlinks* by means of the similarly working *lexical signatures* and suggested a heuristic technique for calculating the signatures. Specifically, their method involves using terms that are rare in the web (have a high inverse document frequency—IDF), while also favoring terms with a high term frequency (TF) within the document, capping TF at 5 to avoid diluting a term's rarity. The IDF of each term is derived from a search engine, while the rest of the signature calculation can be performed locally. Park *et al.* (2002) expanded on this idea by evaluating four basic and four hybrid lexical signature selection methods based on the TF, document frequency—DF, and IDF of those terms. For example, one of their proposed methods TFIDF3DF2 involves selecting two words based on increasing DF order, filtering out words having a DF value of one, and selecting three additional words maximizing TIFF. An important contribution of their work is an evaluation of the documents that the search engine returns in response to a lexical signature query in terms of uniqueness, appearance of the desired document at the top of the result list, and relevance of any other document links returned.

Our approach differs from the two methods we outlined in that it uses the search engine as an oracle for evaluating the selected word set. A stochastic algorithm can rapidly explore the search space (word combinations) to locate the ones that better suit a selection criterion. This allows us, instead of having a fixed algorithm (such TFIDF3DF2) identifying a document's discriminants, to flexibly select from each document the discriminant that maximizes an objective function. Although the objective function we used is based on uniqueness and URL length, different functions such as the relevance of the returned documents could also be used.

3.1 Discriminant Calculation

Trying all document's term combinations to find a unique discriminant is a futilely expensive exercise. The number of n different terms that can be selected from a document is given by

$$\sum_{i=1}^{n} {}_{n}\mathrm{C}_{i} = \sum_{i=1}^{n} \frac{n!}{i!(n-i)!} = 2^n - 1$$

At 31 average unique terms per document of our data set this gives us $4 \cdot 10^9$ different term combinations for each document. Although selecting fewer terms (in practice we found that unique discriminants consisted on average of 1.47 terms) lowers the above figure, the complexity's exponential nature makes the exhaustive search prohibitively expensive on larger documents; selecting 2 out of 1000 terms results in ${}_{1000}\mathrm{C}_2 = 499,500$ combinations (out of the total 10^{30} possible ones).

An efficient deterministic solution to the problem would be preferable to the exhaustive search we outlined above. However, as we will show in the following paragraphs, the problem is intractable, NP-complete.

3.2 Intractability Proof

We will prove that the problem of locating a discriminant that identifies k documents is NP-complete by demonstrating that a tractable P-time solution to the problem could be used to solve the *subset sum* problem, known to be NP-complete (Garey & Johnson, 1979). We can formally express our problem as follows: Let $D = \{T_1, \ldots, T_n\}$ (our document) be a set of terms T. Each term T_i is expressed as a set of the documents it occurs in. Let $D' \subseteq D$ be a document's discriminant. The number of documents k that the discriminant D' with $m = |D'|$ identifies is expressed as the cardinality of the intersection of the corresponding sets:

$$k = \left| \bigcap_{i=1}^{m} D'_i \right|$$

A unique discriminant is one for which $k = 1$.

Similarly, the subset sum problem can be expressed as follows: Let $A = \{a_1, \ldots, a_n\}$ be a set of positive integers. Given an integer s find a set $A' \subseteq A$ with $m = |A'|$ so that

$$s = \sum_{i=1}^{m} a_i$$

If the n elements of the set A have values $0 \ldots m$ we can solve the subset sum problem in terms of the discriminant cardinality problem by using set intersection in the place of addition. For each element $a_i \in A$ we construct a set

$$B_i = \left\{ \begin{array}{ccc} b_{0,1} & & \\ & \ddots & \\ & & b_{m,n} \end{array} \right\} - \left\{ \begin{array}{c} b_{0,i} \\ \vdots \\ b_{m-a_i,i} \end{array} \right\}$$

and let $B = \{B_1 \ldots B_i\}$. The sets we constructed have the property that given a set of positive integers $Q = \{q \mid q \in \{1 \ldots n\}\}$

$$\left| \bigcap_{i=1}^{|Q|} B_{q_i} \right| = \sum_{i=1}^{|Q|} a_{q_i}$$

Using our hypothetical discriminant cardinality P-time algorithm we find a $B' \subseteq B$ such that

$$s = \left| \bigcap_{i=1}^{n} B'_i \right|$$

The a_i elements that correspond to the B' subset will satisfy the subset sum problem

$$s = \sum_{i=1}^{n} a_i \mid a_i \in A'$$

Having shown that the subset sum problem—known to be NP-complete—can be reduced to the discriminant cardinality problem we have proved that the discriminant cardinality problem is also NP-complete.

3.3 Genetic Algorithm

We therefore use a non-deterministic, stochastic algorithm to search the term space. Genetic algorithms (GAs) (Holland, 1975; Goldberg, 1989; Forrest, 1996) are global optimization techniques that avoid many of the shortcomings exhibited by local search techniques on difficult search spaces, such as our unique discriminant selection problem. Goldberg (1994) describes a number of diverse GA applications, while Karr (1993) presents their use for modeling, design, and process control. GAs rely on modeling the problem as a population of organisms. Every organism represents a possible valid solution to the problem. Organisms are composed of *alleles* representing parts of a given solution. Standard genetic recombination operators are used to create new organisms out of existing ones by combining alleles of the existing organisms. In addition, mutations can randomly change the composition of existing organisms. Typically, the algorithm evaluates all the organisms of the population and creates new organisms by combining existing ones based on their fitness. This procedure is repeated until the variance of the population reaches a predefined minimum value or another heuristic criterion is satisfied.

GAs base their operation on a *fitness function* that evaluates an organism's suitability. The fitness function $\overline{O}(x)$ we want to maximize depends inversely on the number of documents N_D a particular discriminant x identifies and, less, on its length L as determined by the number of words N_W and the length of each word L_i:

$$\frac{1}{O(x)} = 100\,(N_D(x) - 1) + \sum_{i=1}^{N_W(x)} L_i + 1$$

As an exception to the above function definition, organisms that fail to identify a single document ($N_D(X) = 0$) are given a fitness rank of 0.

An important characteristic of a genetic algorithm's implementation concerns the representation of each candidate solution. A good representation should ensure that the application of standard crossover recombination operators (where a new organism is composed from parts of two existing ones) will result in a valid new representation. The first organism implementation we used was an ordered set of terms. Thus, for a document containing the words $[ABCDEF]$ two organisms could be $[ACF]$ and $[BE]$. Following experimentation, we found that a boolean vector sized to represent all possible terms of a document—with discriminant terms represented by true values—was a more efficient implementation allowing our code to function in a third of the original runtime. Using a boolean vector scheme, the two above organisms would now be represented as $[TFTFFT]$ and $[FTFFTF]$.

Using the integers 0 and 1 for representing the *true* and *false* boolean values, the genetic algorithm for selecting the minimal unique discriminant out of N different terms can be described in the following steps:

1. [Initialize a population of size S.] Set $P_{0\ldots S,0\ldots N} \leftarrow \lfloor \text{rand}[0\ldots 1] \rfloor$.

2. [Evaluate population members creating the organism fitness vector $\underline{T}$.] For $i \leftarrow 0\ldots S$: set $T_i \leftarrow O(\underline{P_i})$.

3. [Create roulette selection probability vector $\underline{R}$.] Set $R_i \leftarrow \sum_{j=0}^{i}(T_j / \sum_{k=0}^{S} T_k)$.

4. [Create new population using crossovers from the previous population.] For $i \leftarrow 0\ldots S$: select q and r using the roulette selection probability vector so that $R_q \leq \text{rand}[0\ldots 1) < R_{q+1}$ and $R_r \leq \text{rand}[0\ldots 1) < R_{r+1}$, if $\text{rand}[0\ldots 1) <$ crossover rate, set $c \leftarrow \lfloor \text{rand}[0\ldots N) \rfloor$, set $P'_{i,0\ldots c} \leftarrow P_{R_q,0\ldots c}$, set $P'_{i,c+1\ldots N} \leftarrow P_{R_r,c+1\ldots N}$; otherwise set $\underline{P'_i} \leftarrow \underline{P_{R_q}}$.

5. [Introduce mutations.] For $i \leftarrow 0\ldots S$: for $j \leftarrow 0\ldots N$: if $\text{rand}[0\ldots 1) <$ mutation rate, set $P'_{i,j} \leftarrow \lfloor \text{rand}[0\ldots 1] \rfloor$.

6. [Keep fittest organism for elitist selection strategy.] Select f so that $T_f \geq T_{0\ldots S}$, set $\underline{P'_{\lfloor \text{rand}[0\ldots S) \rfloor}} \leftarrow \underline{P_f}$.

7. [Make new population the current population.] Set $\underline{P} \leftarrow \underline{P'}$.

8. [Loop based on the population's variance.] If $\sum_{i=0}^{P} |T_f - T_i| >$ minimum variance go to step 2; otherwise the algorithm terminates with the optimal document discriminant in $\underline{P_f}$.

The implementation of genetic algorithms can be tuned using a number of different parameters. In our implementation we used the parameters that Grefenstette (1986) derived using meta-search techniques namely:

- a population size S of 50,
- a crossover rate of 0.6,
- a mutation rate of 10^{-4},
- a generation gap of 1 (the entire population is replaced during each generation),
- no scaling window, and
- an *elitist selection strategy* (the organism with the best performance survives intact into the next generation).

The random floating point numbers $0 < R < 1$ used for selecting the crossover points, the mutation rates, and the selection of organisms were produced using the *subtractive method* algorithm (Knuth, 1981, 171–173).

In addition we used some domain-specific heuristics:

- we select candidate terms from a subset of the terms with the lowest frequencies across the complete document collection,
- we bias a term's selection according to its global frequency, and
- we ensure that the globally most frequently used terms are not used as candidates.

All the above heuristics are based on the premise that less frequently used terms are more likely to be part of a unique document discriminant.

4 Prototype Implementation

We implemented a set of programs that process a (presumably all-encompassing) set of web pages, and calculate for each page a minimal set of search terms (words) that can be used to uniquely identify that page within the set. To test our implementation we took advantage of the data set provided during the 2002 Google search engine (Brin & Page, 1998) programming contest. The package provided to the contest participants included a programming framework for processing a pre-parsed document collection (the so called *ripper* program) and a large collection of web documents. The breadth and architecture of the *ripper* programming framework strongly indicate that applications based on it could be easily ported to run on the actual Google infrastructure. The 5.9 GB data set we used consists of 916,429 pre-parsed HTML documents containing about $28 \cdot 10^6$ terms.

4.1 Implementation Overview

We calculate the discriminants in two steps:

1. We create an index of all documents where each term occurs; in actual practice a search engine will always have this data structure at hand.

2. For each document we use the genetic algorithm to try combinations of the terms it contains until we find one that does not occur in another document. The processing relies on the index calculated by the previous phase.

As usual the devil is in the details, especially when dealing with the 1 million documents we processed and the $3 \cdot 10^9$ documents currently indexed by typical search engines.

Our application consists of tools for calculating unique discriminants on a single node (useful for proving the concept, trying out the data subset, and experimenting with the algorithms), and in an environment of multiple nodes (for processing the data set of a commercial search engine). In addition, we implemented simple tools for querying the results and obtaining the corresponding URLs.

4.2 Indexing

The index of a large text collection will not typically fit into a computer's fast main memory, while a disk-based structure will probably prove too slow for a realistic application scenario. Some applications have dealt with the problem by compressing the data in-memory (Moffat, 1992; Spinellis, 1994), but this approach would still not accommodate our problem's scope and resource constraints.

We handled the problem by accumulating a term-to-document index in memory and monitoring the memory subsystem's performance. Once the system begins to persistently page (indicating that the memory's capacity has been reached and performance will rapidly degrade due to thrashing), the index is flushed to disk as a sorted file. When all documents have been processed, the partial results are merged into a single file (`idxdata`) containing terms and documents where each term occurs. An index file (`idxdata.idx`) allows rapid serial access to individual terms without having to traverse a term's document list. In addition, a separate file (`idxdata.hash`) allows rapid hash-code based access to individual terms. As the string hash function, we use the one recently proposed for very large collections by Zobel (2001). The hash file is created after the merge phase and can thus be optimally sized, using the Rabin-Miller prime number probabilistic algorithm (Schneier, 1996, pp. 259–260), to minimize collisions.

4.3 Stand-alone Operation

The data-flow diagram of our system's stand-alone operation appears in Figure 1. A new handler of the Google ripper named *index* reads-in preparsed documents and creates (in stages by merging intermediate results) an index of the documents where each term resides (`idxdata`) and two files for accessing that index (`idxdata.idx` and `idxdata.hash`). A fourth file, `topnodes` is set to contain the frequencies of the 100,000 most frequently used terms; it is used as a cache during the discriminant calculation phase.

The second phase is also implemented as a separate ripper handler named *bookmark*. This reads the terms of each document, selects the least frequently used ones, and creates the `bookmarks` file containing for each document its URL and the set of terms forming its unique discriminant. The `bookmark.idx` file created allows

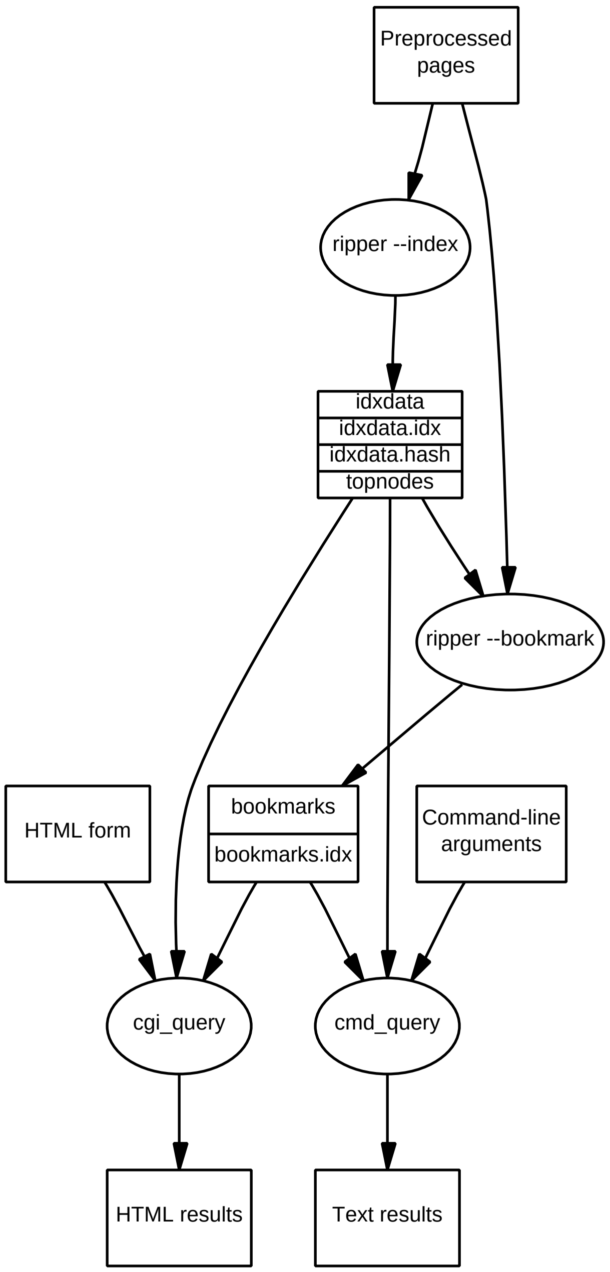

Figure 1: Data-flow diagram of the stand-alone operation.

the location of a given document URL and discriminant based on the corresponding document identifier.

Two query programs we wrote, `cgi_query` and `cmd_query`, will search the term index against the conjunction of a set of specified words and display as search results the matching document URLs and the corresponding unique discriminants. The CGI application presents the discriminants as a URL; the end-user can bookmark this URL and thus visit our search engine to locate the document in the future. The HTML results also contain a link to a Google search with the same discriminant as a search expression. This search will not in general provide unique results since Google's indexed collection is three orders of magnitude larger than the one we processed. One of the discriminants we tried did however identify and correctly locate one page that was moved (renamed): a Google search for the page http://www.umass.edu/research/-ogca/new.htm uniquely identified through the discriminant "ogca fy02" now yields http://www.umass.edu/research/ogca/news/oldnews.htm.

4.4 Distributed Operation

As is apparent from Figure 2, the system's distributed operation is a lot more complicated than the stand-alone case. It does however provide a framework for creating discriminants for orders of magnitude larger collections using a large number of commodity processing nodes. The work distribution strategy is based on two premises:

1. Documents are uniformly distributed across all processing nodes. Each node calculates and serves the discriminants for its documents.

2. Each node is assigned a consecutive subset of terms (e.g. *barometer–beholding*). It is responsible for serving queries (documents that contain a given term) for the terms it is assigned.

To divide the term load across the nodes, we run a stand-alone instance of `ripper --index` on a small representative subset of documents. A separate text file contains a list of all processing nodes. The program `divide`:

- examines the term index and divides it uniformly across processing nodes,

- assigns a separate numeric initial document-id to each node, and

- copies the generated files to all nodes.

On each node we then run an instance of `ripper --index` to process the node-specific preprocessed pages. The program `scatter` is then run on each node to split and copy the resulting term index according to the terms assigned to each node. Each node will thus receive its share of terms as indexed by all other nodes. The `make_index` program merges the node-specific terms generated by all nodes into a single unified index file for the given node. This file is accessed by the node's `index_server` program to provide term document occurrences to other nodes. The `ripper_distr` program run on each node communicates with the `index_server` responsible for a given term to obtain the global list of a term's documents. To reduce network communication overhead initial term frequencies (our algorithm uses a subset of a document's least frequently occurring terms for selecting the unique discriminant)

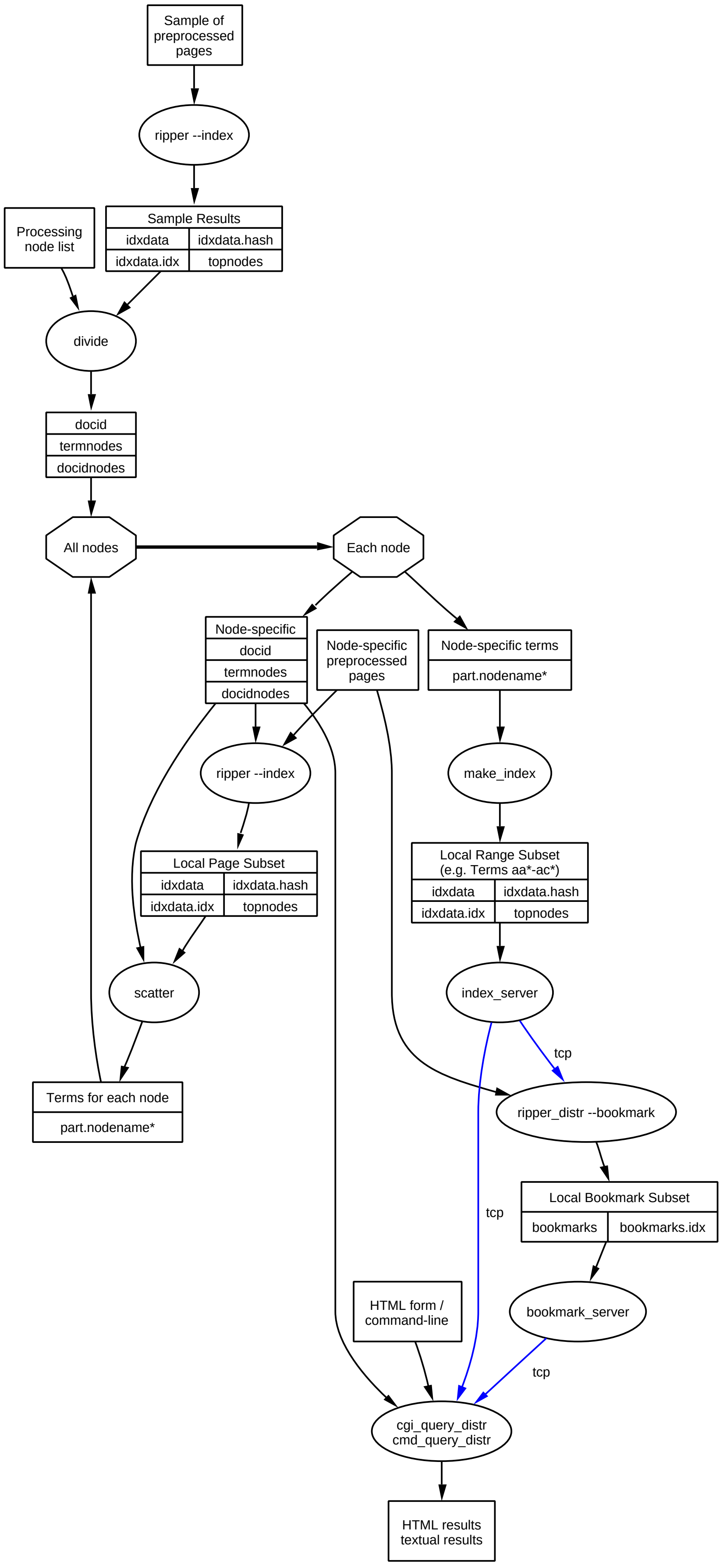

Figure 2: Data-flow diagram of the distributed operation.

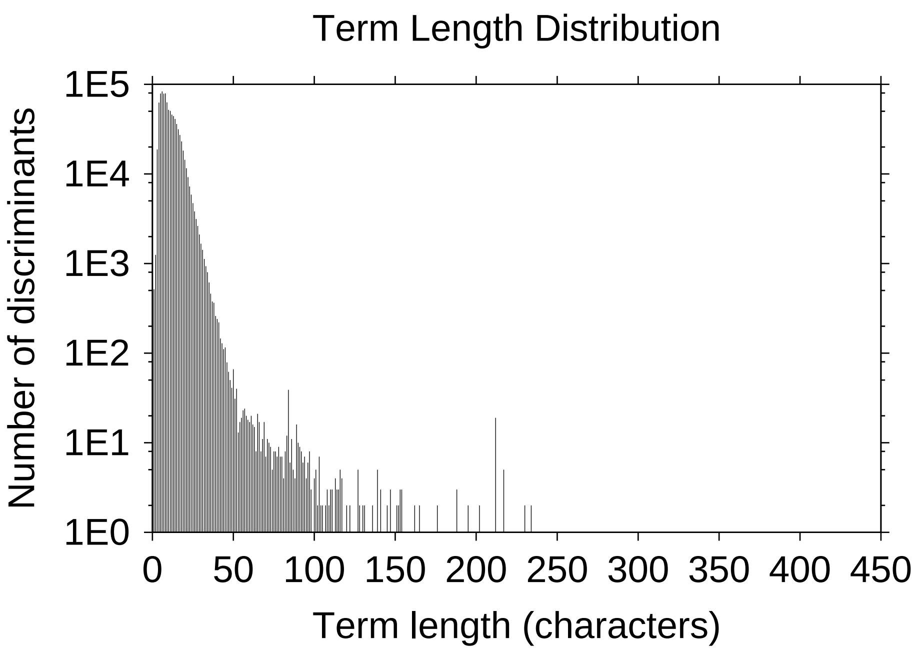

Figure 3: Discriminant term length distribution

are obtained from the local term file; we assume that the local term frequency distribution corresponds to the global one. The generated unique document discriminants can then be accessed by the distributed versions of the query programs by using a document's identification number for locating the node where the respective discriminant is used.

5 Evaluation

Important aspects of algorithms expected to process the global web include, apart from the quality of the results, their requirements on processing time and space, and their scalability.

5.1 Discriminant Performance

After processing the Google sample document collection we found that we needed an average of 1.47 terms to uniquely identify a document. The average length of each discriminant (Figure 3) was 10.77 characters which, as a document identifier, compares extremely favorably with the 43.9 characters of the collection's average document URL length (excluding the initial `http://`. The number of terms for each document's discriminant was distributed as shown in Table 1. The algorithm failed to identify a discriminant for less than 1% of the documents processed.

The property of the calculated discriminants to uniquely identify a document, while not absolute, was we believe acceptable for its intended uses. About 50% of the discriminants our system calculated will locate a single document, while another 10% identify two documents. In total 76% of the discriminants will locate less than 10

Number of terms	# documents	Document %
0 (no discriminant found)	5,107	0.56
1	531,713	58.02
2	331,638	36.19
3	41,132	4.49
4	5,956	0.65
5	783	0.08
6	80	0.01
7	16	0.00
8	3	0.00

Table 1: Discriminant term number distribution.

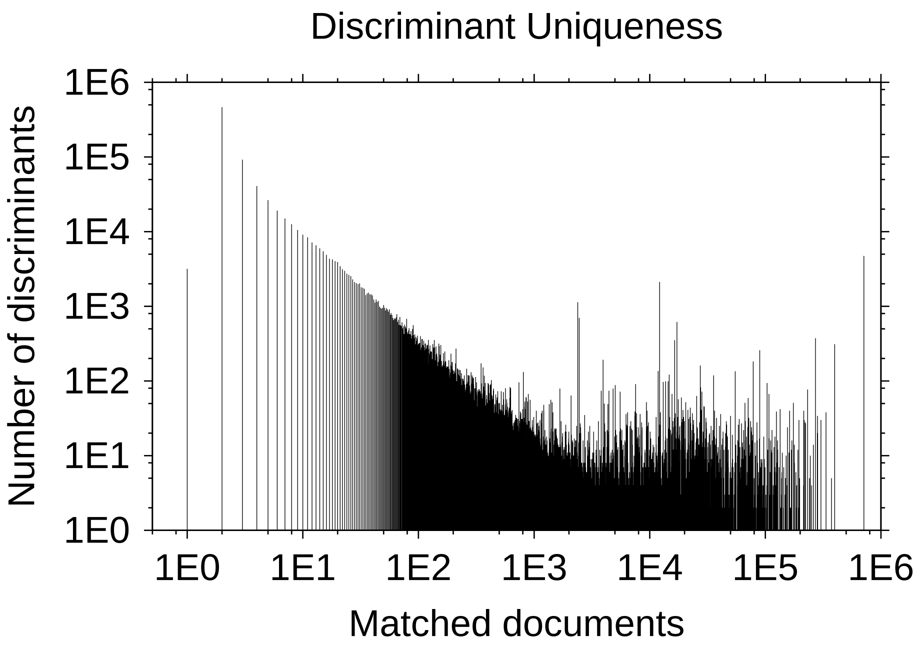

Figure 4: Distribution of document matches across the calculated discriminants.

documents in the sample document collection (Figure 4). These figures can be further improved by tuning various GA parameters such as the number of terms of the candidate set, the number of common terms to eliminate, and the size of the organism pool. Many of the pages for which a unique identifier was not calculated contain very little textual material. As an example our system's spectacular failure to create a unique identifier for the page http://humanities.uchicago.edu/depts/maph/ (it was identified by the term 'humanities', which also matches another 29,497 pages) can be easily explained by the fact that this home page consisted entirely of text and pictures presented in graphical hypertext form using image maps.

5.2 Algorithm Performance

On average the GA was run for 10.6 generations to calculate each discriminant. However, the distribution of the GA generations that were required was highly skewed: the corresponding mode was 2, median 5, and the standard deviation 16.5. To evaluate the performance of the GA over the heuristic selection of words, we calculated discriminants for a subset of $13,000$ documents using a procedure that followed the selection traits of the GA, but not its evolutionary strategy. Specifically, for every document we created $S = 50$ organisms and let those mutate G times, where G was the number of generations the GA had run for that document. The initial organism was not random, but as was the case for the GA, was created using the allele probability selection vector. The results from this quasi-random selection were 34% worse than those obtained from the GA operation. In 58% of the cases the two methods yielded the same result. Although the advantage offered by the GA may not seem impressive, we believe that (a) only the GA will scale to handle the three orders of magnitude larger collection of the global web, and, (b) the GA can be easily adapted to work with more demanding objective functions, whereas the heuristic techniques can not.

5.3 Time Requirements

We indexed the sample document collection (916,429 documents 3,152,415 unique lowercased terms) on a 733MHz, Celeron CPU, 128MB RAM, 40GB IDE disk machine in 18,605 real, 7,064 user, and 1,533 system seconds, at a throughput of 29.26 documents / s. The process used 18 intermediate indexed files each of about 75MB in size (the first one was 190MB) with an intermediate file being dumped every six minutes. The hashed term database performed adequately, but not spectacularly with 2,111,778 total term name collisions (resulting on an average of 1.49 disk index accesses per term) and 173 maximum term name collisions.

The time to perform the unique discriminant calculations varied, because we split the workload among 20 different machines. The time required on 733MHz, Celeron CPU, IDE disk machine for processing the file `pprepos.00` (16,564 documents) was 58,721 real, 184,054 user, and 9,604 system seconds giving a processing time of 3.54 s / document. Systems with a SCSI disk subsystem performed better.

We unfortunately lacked appropriate resources (a large farm of networked processing nodes of similar technical characteristics) to perform rigorous experiments on the distributed implementation of our system. We were however able to obtain a lower bound of the expected performance by running the programs on a small number of workstations. By extrapolating from our results, we calculated the expected distributed operation time T' given a standalone time of T as $T' \geq T \times 2.2$.

5.4 Space Requirements

The indexing operation utilizes the maximum amount of main memory available, but is constrained by design to stop its memory usage growth once thrashing occurs. In our case, it processed without a problem the complete sample data set on machines with 128MB RAM. The off-line space requirements are comprised of the space needed to store the term index and its hash file; this was for the sample document collection 826,504,382 bytes for the index and 50,438,784 bytes for the hash file, giving an overhead of approximately 957 bytes per document.

The space requirements of the discriminant calculation phase are more difficult to judge. When performing calculations over the sample collection we observed a maximum resident set size 62,140KB. This number is likely to grow with a larger document set, but not by much, since it reflects the space needed to store the document instances of a document's least frequent terms. Given that prior to the candidate set selection, only term frequencies are stored in memory, the term selection process can be easily adjusted to dynamically select terms that will load in the main memory a fixed number of document occurrences, thereby providing a concrete bound to the memory usage.

5.5 Scalability

Will our approach gracefully scale to cover a search engine's complete page collection? We consider as a test case Google's $2 \cdot 10^9$ page collection and for our estimates we use a number of 8,000 processing nodes reported in the technical press as comprising Google's infrastructure (Wagner, 2001).

Computing the term index will therefore require:

$$\frac{2 \cdot 10^9 \times 0.02}{8,000}\text{s} = 83\text{minutes}$$

The communications overhead will be roughly equivalent to that of copying the index files across the network; given an index file size of

$$11,000 \times \frac{2 \cdot 10^9}{8,000} = 2.75 \cdot 10^9\text{bytes}$$

this will add an overhead of 8 minutes using a switched 100Mbit network with a 50% utilization rate. Merging the intermediate files is unlikely to present a problem; at one point an error in our thrash monitoring code resulted in 1,700 intermediate files which were merged without a problem.

The time to compute the discriminants will be larger. At 3.54 s / document the calculation of all discriminants will require

$$3.54 \times \frac{2 \cdot 10^9}{8,000} = 885,000\text{s} = 10\text{days}$$

One should keep in mind that this process will be required to run very infrequently for the entire document collection and can then be run incrementally as new documents are added (the entire point of the unique discriminants is that they remain valid with a very high probability even as the structure of the web changes).

Note that all our time figures are based on the results we obtained using low-end 733MHz Celeron PCs with IDE hard disks. In addition, the indexing phase can be omitted and the discriminant calculation phase can be easily adjusted to use a search engine's existing term index structure. The discriminant calculation algorithm only needs access to an oracle that answers the question of how many documents are matched by a given term combination. We assume that a search engine's infrastructure is engineered and tuned to efficiently answer the above query and should therefore preferably be used by our algorithm.

6 Conclusions and Further Work

In the previous sections we outlined how our application utilizes search engine technologies to address an important shortcoming of today's web in a scalable, and efficient manner. The alternative document identifiers we calculate are not only resilient to URL changes, but also almost a fourth of the size of conventional URLs.

However, the document identifiers we provide are not suitable as universal replacements for the URLs currently employed. In document retrieval terms their use involves a trade-off of noise over silence. Our calculation method is not perfect and there are cases where our algorithms will either fail to calculate a discriminant for a web page (e.g. when the page does not contain any text), or will calculate a discriminant that will match multiple pages. In addition, changes to the web contents and a search engine's coverage can make identifiers that were calculated to be unique match additional documents. Although we have not studied the temporal behavior of the discriminants we calculate, an obvious pathological case involves documents cloned from a given source document through small additions or changes. This cloning is a common operation and is related to the scale-free topology of the web (Barabási *et al.*, 2000). Cloned documents will very probably match the discriminant calculated for their original ancestor. Based on the above, we believe that unique discriminants are most suited for situations where a human can make an informed choice for using them and remains in the loop during their use. As an example, personal bookmarks are particularly well suited for being stored using unique discriminants (or include unique discriminants as a fail over mechanism). Bookmarks are highly prone to aging since they are typically not formally maintained; in addition, once a bookmark matching several documents is followed, the user can intelligently choose between the different pages.

During our work, we noted a number of improvements that could be employed for optimizing the algorithm's performance and for further increasing the usefulness of the obtained results. These include:

- Optimize the GA, tuning its configuration by replacing the generic parameters we used with parameters selected for the given problem and the properties of the web. In an examination of the discriminants we calculated, we found some instances of duplicate discriminants and multiple documents identified by the same discriminant that could have been easily avoided.

- Experiment with different stochastic algorithms such as simulated annealing (Cerny, 1985; Van Laarhoven & Aarts, 1987; Koulamas *et al.*, 1994) and tabu search techniques (Glover, 1990).

- Explore the possibility of using our results for locating duplicate documents. Since our algorithm tries very hard to find unique discriminants, failure to find unique discriminants could signify the existence of virtually duplicate documents. This technique can be strengthened by resetting the random number generator seed values before processing each document.

- Investigate the impact of cloning and modification on the temporal effectiveness of discriminants, experimenting with different objective functions based on the notion of similarity and search engine ranking.

- Study and improve the distributed algorithm operation on a large network of hosts.

- Associate with each URL and discriminant a unique multi-byte hash code that will accurately identify the precise location of pages that have not moved.

We end our description, by noting how the realization of the application we outlined was made possible only through the combination of multiple computer science disciplines: information retrieval was the domain where our problem was formulated, algorithms and data structures provided the framework to obtain the solution, operating system concepts allowed us to bound the indexing memory requirements, complexity theory gave us the theoretical background for searching for algorithmic solutions, stochastic approaches were used for sidestepping the problem's NPC characteristics, and networking and distributed systems technology provided the framework for developing the distributed implementation. Increasingly, the immense scale of the web is necessitating the use of multidisciplinary approaches to tackle information retrieval problems.

Acknowledgements

During the system's implementation phase I benefited from fruitful discussions with Elisa Fragkaki, Stavros Grigorakakis, Vasilis Kapouleas, and Michalis Vazirgiannis. The string hashing expression used in the hashing functor was written by Justin Zobel. The prime number generator used for sizing the hashed database contains code developed by Bradley Smith and Greg Heileman at the University of New Mexico. The distributed version of the system relies on the *socket++* library developed by Gnanasekaran Swaminathan and patched for Linux and FreeBSD by Lauri Nurmi. Finally, I would like to thank the paper's anonymous referees for their insightful and astute comments on an earlier version of this paper.

References

Ashman, H., Electronic Document Addressing: Dealing with Change, *ACM Computing Surveys*, **32**(3), 201–212 (2000).

Barabási, A.-L., Albert, R., & Jeong, H., Scale-free characteristics of random networks: the topology of the world-wide web, *Physica*, **A**(281), 69–77 (2000).

Berners-Lee, T., Masinter, L., & McCahill, M., *RFC 1738: Uniform Resource Locators (URL)*, 1994 (Dec.). Updated by RFC1808, RFC2368 (Fielding, 1995; Hoffman *et al.*, 1998). Status: PROPOSED STANDARD.

Brin, S., & Page, L., The Anatomy of a Large-Scale Hypertextual Web Search Engine, *Computer Networks*, **30**(1–7), 107–117 (1998), Seventh International World Wide Web Conference Proceedings (WWW7).

Cerny, V., Thermodynamical Approach to the Traveling Salesman Problem: an Efficient Simulation Algorithm, *Journal of Optimization Theory and Applications*, **45**, 41–51 (1985).

Chankhunthod, A., Danzig, P. B., Neerdaels, C., Schwartz, M. F., & Worrell, K. J., A Hierarchical Internet Object Cache, *in USENIX Technical Conference Proceedings*, Usenix Association, Berkeley, CA, 1996.

Fielding, R., *RFC 1808: Relative Uniform Resource Locators*, 1995 (June). Updates RFC1738 (Berners-Lee *et al.*, 1994). Updated by RFC2368 (Hoffman *et al.*, 1998). Status: PROPOSED STANDARD.

Forrest, S., Genetic Algorithms, *ACM Computing Surveys*, **28**(1), 77–83 (1996).

Garey, M. R., & Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, 1979.

Glover, F., Tabu Search — Part I, *ORSA Journal on Computing*, **1**, 190–206 (1990).

Goldberg, D. E., *Genetic Algorithms: In Search of Optimization & Machine Learning*, Addison-Wesley, 1989.

Goldberg, D. E., Genetic and Evolutionary Algorithms Come of Age, *Communications of the ACM*, **37**(3), 113–119 (1994).

Grefenstette, J. J., Optimization of Control Parameters for Genetic Algorithms, *IEEE Transactions on Systems, Man, and Cybernetics*, **16**(1), 122–128 (1986).

Hitchcock, S., Carr, L., Harris, S., Hey, J. M. N., & Hall, W., Citation linking: improving access to online journals, *pages 115–122 of Proceedings of the 2nd ACM international conference on Digital libraries*, 1999.

Hoffman, P., Masinter, L., & Zawinski, J., *RFC 2368: The mailto URL scheme*, 1998 (July). Updates RFC1738, RFC1808 (Berners-Lee *et al.*, 1994; Fielding, 1995). Status: PROPOSED STANDARD.

Holland, J. H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Michigan, 1975.

Karr, C. L., Genetic Algorithms for Modelling, Design, and Process Control, *pages 233–238 of CIKM '93. Proceedings of the Second International Conference on Information and Knowledge Management*, ACM, 1993.

Knuth, D. E., *The Art of Computer Programming*, second edn., Vol. 2: Seminumerical Algorithms, Addison-Wesley, Reading, MA, 1981.

Koulamas, C., Antony, S. R., & Jaen, R., A Survey of Simulated Annealing Applications to Operations Research Problems, *Omega International Journal of Management Science*, **22**(1), 41–56 (1994).

Lawrence, S., & Giles, C. L., Searching the Web: General and Scientific Information Access, *IEEE Communications*, **37**(1), 116–122 (1999).

Lawrence, S., Giles, C. L., & Bollacker, K., Digital Libraries and Autonomous Citation Indexing, *IEEE Computer*, **32**(6), 67–71 (1999).

Lawrence, S., Pennock, D. M., Flake, G. W., Coetzee, F. M., Glover, E., Nielsen, F. Å., Kruger, A., & Giles, C. L., Persistence of Web References in Scientific Research, *IEEE Computer*, **34**(2), 26–31 (2001).

Moffat, A., Economical Inversion of Large Text Files, *Computing Systems*, **5**(2), 125–139 (1992).

Park, S.-T., Pennock, D., Giles, L., & Krovetz, R., Analysis of Lexical Signatures for Finding Lost or Related Documents, *pages 11–18 of Proceedings of the 25th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, New York: ACM Press, for ACM, 2002.

Phelps, T. A., & Wilensky, R., Robust Hyperlinks: Cheap, Everywhere, Now, *in Proceedings of Digital Documents and Electronic Publishing (DDEP00)*, 2000.

Pitkow, J. E., Summary of WWW Characterizations, *World Wide Web*, **2**(1–2), 3–13 (1999).

Schneier, B., *Applied Cryptography*, second edn., Wiley, New York, 1996.

Spinellis, D., The Design and Implementation of a Legal Text Database, *pages 339–348 of* Karagiannis, D. (ed), *DEXA 94: 5th International Conference on Database and Expert Systems Applications*, Springer-Verlag, 1994. Lecture Notes in Computer Science 856.

Spinellis, D., The Decay and Failures of Web References, *Communications of the ACM*, **46**(1), 71–77 (2003).

Takeda, M. K. K., Information Retrieval on the Web, *ACM Computing Surveys*, **32**(2), 144–173 (2000).

Van Laarhoven, P. J. M., & Aarts, E. H. L., *Simulated Annealing: Theory and Applications*, D. Reidel, Dordrecht, The Nethelands, 1987.

Wagner, M., Google Defies Dot-com Downturn, *TechWeb*, Apr. (2001), Online http://www.techweb.com/wire/story/TWB20010427S0011 (current June 2002).

Zobel, J., Heinz, S., & Williams, H. E., In-memory Hash Tables for Accumulating Text Vocabularies, *Information Processing Letters*, **80**(6), 271–277 (2001).