Topic Distillation in Desktop Search

Alex Penev, Matthew Gebski, and Raymond K. Wong

[1] National ICT Australia, Sydney, NSW 2052, Australia
[2] School of Computer Science and Engineering
The University of New South Wales, Sydney, NSW 2052, Australia
{alexpenev, francg, wong}@cse.unsw.edu.au

Abstract. Desktop Search, the search across local storage such as a personal computer, is a common practice among computer users. There has been much activity in Web-related Information Retrieval, but Desktop Search has only recently increased in popularity. As the structure and accessibility of data in a local environment is different to the Web, new algorithmic possibilities arise for Desktop Search.

We apply a connectivity analysis approach to the local environment—a filesystem. We describe how it can be used in parallel with existing tools to provide "more useful" ranked results. Our evaluation reveals that such an approach has promise, and we conclude that exploiting the organization of a filesystem is beneficial for Desktop Search.

1 Introduction

Desktop Search (DS) refers to searching of local, contained data stores, as opposed to searching a foreign and overwhelming repository such as the Internet. There are often many (and unfamiliar) documents that satisfy an Internet query, but users generally seek specific (and familiar) files on a desktop.

DS tools place emphasis on using an inverted file to facilitate near-real-time retrieval, by indexing the content and metadata of various known filetypes. One of the first such tools, Lotus Magellan (late 1980s), created a master index of different filetypes and allowed searching and viewing of files without the need to launch the associated viewer application. Such functionality is emulated by today's DS tools.

The field received more research attention following the prototype work of [1]. AskJeeves, Google, Microsoft and Yahoo have all since released DS tools, and there are currently more than twenty available[1]. By applying the inverted file index to local data, DS tools produce sub-second searches.

Inverted files substantially improve conventional disk-scanning methods, but only address the problem of speed. Discerning the relevance of results requires analyzing their content. Fortunately, simple content analysis is inexpensive when inverted files are involved, and the recent wave of DS tools has been welcomed by many. The tools mainly differ in their presentation, index size overhead, extent of filetype coverage, and brand power.

[1] http://www.goebelgroup.com/desktopmatrix.htm, visited Dec 2005.

S. Bressan, J. Küng, and R. Wagner (Eds.): DEXA 2006, LNCS 4080, pp. 478–488, 2006.

"Topic distillation" refers to educing high-quality documents that are most-representative of a given query topic[2]. Our motivation for this work lies in distilling such high-quality files to improve ranked DS results. For this, we adopt and adapt a connectivity analysis approach.

The remainder of this paper is organized as follows: §2 provides background on DS and connectivity analysis. §3 explains our approach and methodology, while §4 details our evaluation and observations. §5 highlights related work, and we draw conclusions in §6.

2 Background

2.1 Ranking in Desktop Search

Today's DS tools provide options to sort the results by metadata, such as lexically, chronologically, by filesize, or clustered by format. One problem with searching for local data is that the result set can still be large. Without knowing the metadata, sorting by these attributes is unlikely to be useful; the user will need to sequentially scan the result list ($\frac{n}{2}$ inspections on average).

We believe that a ranked list is useful to the user, as they may not always remember metadata (or even know it in the first place, if looking through files not authored by them). A choice to sort by relevance or by metadata will allow the user to pick a strategy they think will result in the fewest inspections. Several of today's DS tools provide such a choice, which supports our view.

Our goal is to improve ranked lists, by placing files that are most-representative of the query topic on top. On the assumption that the user's query adequately specifies their target, finding such representative files is coaxial to the problem of topic distillation. Originating in bibliometrics, connectivity analysis approaches have proven useful in solving this problem for Web IR[3,4].

2.2 Content and Connectivity Analyses

Content Analysis measures how well-representative a document is of a query. Approaches such as the Vector Space Model are popular.

Connectivity Analysis measures how well-connected a node is in a graph. The World Wide Web, a large directed graph, is regularly subjected to such analysis to elicit the popular pages. Two celebrated algorithms in Web-IR are *PageRank* and *Hypertext-Induced Topic Selection* (HITS).

PageRank[3] calculates the probability distribution of random walks, using the random surfer model on a query-independent global graph. Using the in-degree, PageRank deems a page as popular if it has many in-links (and even *more* popular, if the links come from already-popular pages).

HITS[5] argues that every document serves two purposes: to be an authority or to be a hub. An authority is a page with topic-relevant content, while a hub simply links to authorities (its content may be irrelevant). Like PageRank, HITS uses in-links for calculating authoritativeness. But it uses out-links for hubness. Depending on the situation, hubs can be more-useful search results.

Both of these algorithms are iterative, and "shake" the graph to distribute weight along the hyperlink-edges until an equilibrium is reached.

We chose to adopt HITS due to its classification of nodes as authorities and/or hubs, which maps intuitively to a filesystem (§3.1). PageRank lacks this classification, considering nodes as equals. Moreover, HITS uses a query-specific subgraph for its root set and subsequent calculations, allowing us to easily build on top of the retrieved results of an existing DS tool (§4.2).

2.3 HITS

HITS[5] gathers documents from the Web and nominates the best hubs and authorities among them. [6] notes that "a large amount of latent human judgment is encoded within hyperlinks, with links to another page offering some measure of conferred authority". This conferral of authority is used to distill reputable documents. Hubs and authorities exhibit a mutually-reinforcing, bipartite relationship, whereby a good hub should link to good authorities, and a good authority should be linked to by good hubs.

The algorithm begins by retrieving a small *root set* of pages via a search engine, i.e. a query-specific subgraph. Nodes in this set may be highly disconnected, and a *base set* is formed by augmenting the immediate neighborhood of documents that are 1 hop away. The augmented documents add structure to the graph, and may also include relevant pages that did not match the keywords but were frequently cited in the root set[2].

The base set induces a directed graph $G = (V, E)$, with vertices V and edges E representing pages and hyperlinks, respectively. All $v \in V$ are assigned both a hub and authority score, and HITS updates the new scores from the old using two alternating summations until the scores converge:

`HITS`. *in*: link-graph vertices V, directed edges E. *out*: hub/auth score vectors h/a.
1. $\forall v \in V$, initialize $h_0[v]$ and $a_0[v]$ to 1.
2. **for** iteration k **until** convergence:
3. $\quad \forall v \in V, \quad h_k[v] := \sum_{o:(v \to o) \in E} a_{k-1}[o]$
4. $\quad \forall v \in V, \quad a_k[v] := \sum_{i:(i \to v) \in E} h_{k-1}[i]$
5. $\quad$ normalize h_k and a_k.
6. **return** h and a

Algebraically, HITS computes the dominant eigenvectors of the $M^T M$ co-citation and MM^T coupling matrices, where M is the $|V| \times |V|$ non-negative adjacency matrix of the focused subgraph. The algorithm, as well as positively-weighted versions of it, has been shown to converge[5,7] under Perron-Frobenius.

Because HITS considers only the structure (in- and out-links), it is occasionally prone to undesirable behaviors, such as *mutually reinforcing relationships between hosts* (alleviated by reducing the weight of in-links from the same host[7]), and *topic drift* (alleviated by using content analysis to maintain focus on the query[7,8,5]).

[2] A classic example is that search engines are linked to by pages talking about "search engines", but rarely include those two keywords on their page.

3 Approach

While parts of HITS can be adopted to suit a filesystem, other parts need to be adapted. The most important is the concept of connectivity.

3.1 Filesystem Connectivity

On the Web or in research literature, connections/links are created by human judgment. It is evident that Web authors arrange information into directed graphs that link related content together. Unfortunately, a filesystem lacks this concept of *conferral of authority*. In a filesystem, the organization hierarchy involves human judgment, but the implicit links between files/dirs are not created on an individual basis. However, it has been noted[9] that users generally:

- organize files into subdirectories
- group related files together
- group related directories nearby

Therefore, while we lack the luxury of conferred links, we can exploit the organization of the filesystem hierarchy to determine "relevant neighborhoods" that contain relevant directories. We will retrieve *relevant files* (in a content analysis sense), but would prefer relevant files found in *relevant directories*. Since links are largely artificial[3], our notion of a *relevant directory* is not only one that contains relevant files, but also one that resides in a *relevant neighborhood*. The remainder of this section outlines the ways in which we construe HITS to suit this goal.

The concept of *hubs* and *authorities* conveniently map to *directories* and *files*, and by applying the mutually-reinforcing, bipartite relationship that they exhibit on the Web, we draw the analogy that a good directory should contain good files, and a good file should be contained in a good directory. The hub and authority scores tell us exactly this.

3.2 Root and Base Sets

The root set, the query-specific subgraph, can be built using the indexing and searching functionality of a current DS tool. We retrieve the top 250 results using Lucene[10]. To expand this set F of results into the base set, we use the filesystem hierarchy to add the missing structure into our graph: we create a directory set D that forms a spanning tree on F. No new files are added, but the simple injection[4] of directories—which dually serve as hubs—provide paths. Our base set, $D \cup F$, induces a strongly-connected link graph, where every node has a route to every other. We refer to the length of the (shortest) path between two nodes as their **degree-of-separation**, or δ.

This δ is the equivalent of Marchiori's "how many clicks"[11] an end-user needs to perform to get from one page to another, by using only atomic actions.

[3] Placing a new directory anywhere in the filesystem automatically causes unintentional paths (of some length) to many other node in the hierarchy.

[4] E.g. `/home/me/a.txt` will have `a.txt` in F, and `/home/me`, `/home` and the filesystem root `/` in D.

The δ for two directories α and β can be directly calculated from their paths, by finding their lowest common ancestor-or-self directory λ: $\delta(\alpha, \beta) = (|\alpha| - |\lambda|) + (|\beta| - |\lambda|)$, where $|n|$ is depth of n. For files, the immediate parent directory is used. Therefore, by construction, δ is 0 between a directory and itself, a file and itself, or a file and its immediate parent directory.

Similar to [11,12], we use δ as a fading factor to decay the influence of nodes upon each other as they get farther apart. We fade proportional to $\frac{1}{\delta^2}$, a non-linear penalizing curve that is harsh on distance but lenient on proximity. Other functions may be used[5].

3.3 Algorithm

Our `hitsFS` algorithm adapts HITS to our filesystem reinterpretation of what constitutes a link and a relevant hub.

`hitsFS`. *in*: dirs D, files F. *out*: hub scores h for D, authority scores a for F.

1. $\forall d \in D$ and $\forall f \in F$, initialize $h_0[d]$ and $a_0[f]$ to 1.
2. **for** iteration k **from** 1 **to** K:
3. $\quad \forall d \in D, \quad h_k[d] \quad := \sum_{f \in F : \delta(d,f)=0} a_{k-1}[f] \quad + \sum_{d' \in D} \frac{h_{k-1}[d']}{(1 + \delta(d, d'))^2}$
4. $\quad \forall f \in F, \quad a_k[f] \quad := \quad ca(f) \times \sum_{d \in D} \frac{h_{k-1}[d]}{(1 + \delta(d, f))^2}$
5. $\quad$ normalize h_k and a_k.
6. **return** h and a.

Intermediate results for the output vectors h and a during the k-th iteration are labeled as h_k and a_k. Line 3 updates the hub score for each d, by summing the weights of base set files that are in d. We then augment the surrounding neighborhood influence from other hubs (decayed by distance). Line 4 updates the authority scores, whereby a file f is influenced by its content analysis score ($ca(f)$, in [0,1] as returned by Lucene). HITS models an authority's score as the sum of hubs that point to it, which we achieve by summing the influence of other hubs on f (decayed by distance).

In our experiments, we set the number of iterations, K, to 20. We have noticed, however, that about 10 iterations are enough to stabilize the top-most scores—an observation supported by the literature.

4 Evaluation

4.1 Corpus

Unfortunately, existing work uses unsuitable corpora for our task. Link analysts have often focused on the Web, downloading large portions of online domains.

[5] Experimental results were remarkably similar with $2^{-\delta}$; however, we prefer the inverse-square-law, as it has physical interpretations, describing many natural force-distance relationships.

Such corpora are non-static and difficult to procure. As we are only interested in hierarchy and content (not hyperlinks), we used the JDK 1.5.0, a familiar corpus[6] that is widely variant in structure.

This corpus contains Java library code, which we tokenize[7] to remove syntax problems. In the modified corpus, there are 9267 files in 674 directories. There is an average of 13.8 files per dir (max 321, median 5), and 927 tokens per file (max 37812, median 463). The mean directory depth is 4.52 (max 9, median 4).

We index this modified corpus with several DS tools, described in the next section. Although Java classes generally reference other Java classes, we do not use these connections as "links" in any way.

4.2 Competition

We test our prototypes against three of the best[13] DS tools: Copernic Desktop Search, Google Desktop Search v2, and Microsoft's Windows Desktop Search[8]. We refer to these as *CDS*, *GDS*, and *MDS*. Our prototype, DirH, retrieves the top 250 results from Lucene v1.9 [10], and applies `hitsFS`.

We chose Lucene since it reports content analysis scores (VSM) that `hitsFS` needs, although any listing of documents with their similarity score suffices. As `hitsFS` seeks a single numerical representation for the top documents, the underlying metric may be a black-box.

We provide two versions of our prototype—**DirH0** and **DirH1**—which have Lucene and a modified-Lucene as backend. We include both Lucenes in our evaluation (*Luc*, *Lucmod*). The modifications are:

- stemming is enabled.
- document length norm is changed to 1 ($nwords^{-0.5}$ often penalizes longer files on length more than shorter files are penalized for lower TF).
- keywords-matched ratio is squared. Disjunctive matching and stemming significantly increase the number of matches, yet we still only use the top 250 as root set. We wish to doubly-penalize files that match only a few keywords, to promote better hubs by having more files represent them in the top 250.

These tools are summarized in Table 1.

4.3 Queries

Our experiment focused on ranking highly-representative file(s) at the top, for a given query topic. To promote objectivity in our results, the queries and their

[6] http://java.sun.com; specifically, the j2se/src/share/classes tree, without compiling natives.

[7] We delete non-java files, and use javadoc to generate documentation. This step removes the code/implementation, but semantic meaning is retained since comments account for much of the generated documentation. We then remove javadoc-introduced package/tree navigation pages. The extant files are API documentation versions of the original classes, which we strip clean of markup (html2text), tokenize and case-fold to plain-text.

[8] http://www.copernic.com, http://desktop.google.com, http://desktop.msn.com

target answers (Table 2) were taken from reputable Java FAQs[14,15]. However, since users tend to search using 3 terms or less[16,17], we have condensed the queries to at most 3 terms.

Table 1. Summary of DS tools trialled. Our prototypes are in bold.

DS Tool	Ranking?	Stemming?	Grouping?	Stopwords?	Path match?
CDS	No	Weak	AND	Index	Yes
GDS	Yes	No	AND	Index	Yes
MDS	Yes	Weak	AND	Index	Yes
Luc	Yes	No	OR	Ignore	No
DirH0	Yes	No	OR	Ignore	No
Lucmod	Yes	Porter	OR	Ignore	No
DirH1	Yes	Porter	OR	Ignore	No

Table 2. DS tools were subjected to the above queries

Id	Query	Targets
q1	connect remote server	java.net.{Socket, URLConnection}
q2	select file	javax.swing.JFileChooser
q3	call garbage collector	java.lang.System
q4	execute external program	java.lang.Runtime
q5	schedule thread	java.util.{Timer, TimerTask}
q6	make beep sound	java.awt.Toolkit
q7	linked list	java.util.{LinkedList, Vector, ArrayList}
q8	convert strings numbers	java.lang.{Integer, Double, Long, Short, Byte, Float}
q9	write object stream	java.io.Serializable
q10	play sound	java.applet.AudioClip
q11	list supported fonts	java.awt.Toolkit
q12	read data file	java.io.{FileInputStream, FileReader, BufferedReader, DataInputStream}
q13	write data file	java.io.{FileOutputStream, PrintWriter, PrintStream}
q14	append data file	java.io.{FileOutputStream, RandomAccessFile, DataOutputStream}
q15	format numbers	java.text.{NumberFormat, DecimalFormat}
q16	convert ip address	java.net.InetAddress
q17	generate random integer	java.util.Random
q18	display image	javax.swing.ImageIcon, java.awt.Image

4.4 Results

Each tool answered each query, but only the top 50 results were acknowledged. We report two metrics (Table 3): MRR and a Mean *normalized* Reciprocal Rank.

Since very few results are accepted as correct, a hit-and-miss tool can still be moderately competitive under MRR's steep $\frac{1}{r}$ curve[9]. In Fig. 1, we provide

[9] Softer curves (e.g. $\frac{1}{\sqrt{r}}$, $\frac{1}{1+\log r}$ or $\frac{n+1-r}{n}$) place DirH1 in the lead for both metrics.

Table 3. MRR and Mean normalized Reciprocal Rank results

Score	CDS	DirH0	DirH1	GDS	Luc	Lucmod	MDS	Avg.
MRR	0.089	**0.440**	0.416	0.224	0.358	0.305	0.314	0.307
MnormRR	0.087	0.323	**0.425**	0.224	0.272	0.292	0.254	0.268

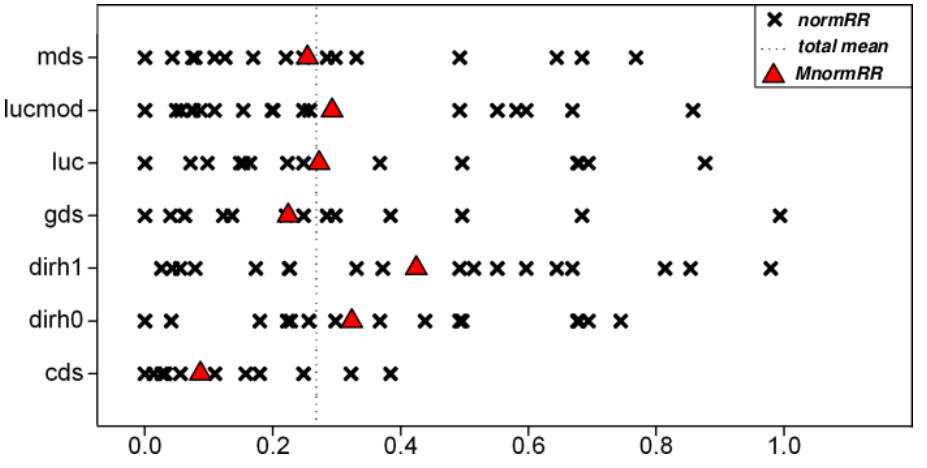

Fig. 1. Mean normalized Reciprocal Rank observations for Table 2 queries. MnormRR vector-normalizes the per-query RR scores to share the weight according to a tool's *relative performance* to the others. This graph shows that most tools are roughly equal and close to the mean, but both of our prototypes performed better on average.

a normalized RR metric, which shares the scores relative to the performance of the other tools. The interpretation of the graph is that our approach *generally ranks a target higher* than its competitors, when using the top 50 results.

The behavior for some queries was not unexpected. Luc often penalized targets on length, so Lucmod ranked them higher. DirH1 used Lucmod's higher content analysis placement to maintain a good ranking for the target.

Q6 was the only query which conjunctive tools failed on, since no file matched all words. Luc matched one term, ranking the target at 183rd. Lucmod matched a second term via stemming, and since it had no length penalty, placed the target in 3rd. DirH1 pushed it down to 5th, since its hub was only ranked 8th.

Several queries had very common terms. DirH1 deemed java.lang the best hub for q8, reporting all but one target (despite 92% of the corpus being matched). DirH0 listed 4 (15% match due to no stemming). Other tools did not report a target in their top 50. Stemming was unnecessary for q9, and all disjunctive tools matched 93% of the corpus. The "noise" from the numerous content-relevant files resulted in only GDS, DirH1 and MDS reporting the target.

DirH1 suffered from polysemy in some queries. It reported all targets for q7, but its top 25 positions were dominated by the sun.tools.doclets.formats.html

package (classes refer to *link* and *list* in an HTML context). With stemming off, DirH0 avoided this problem.

For q7 and q10, the existence of better hubs caused our approach to lower the rank of Luc/Lucmod's content analysis suggested positions of the target. But for the other queries where either tool reported a target in the top 50, our approach placed a target in an equal or better rank.

In terms of performance, total processing time averaged to 2.01s on a 3.2MHz P4 with standard load. Most of this time was used for disk IO in the building of auxiliary data structures, which would normally be bootstrapped only once by a background-process DS tool. Computation of 20 iterations under `hitsFS` averaged to 0.064s, which does not grow with the size of the result set since we use a maximum root set size of 250.

5 Related Work

PageRank and HITS (§2.2, [3,5]) are popular connectivity analysis algorithms, and have had many suggested improvements (for a more-extensive survey, we refer to [6]). Our work builds on HITS by adapting the concepts of what constitutes a hub, an authority, a link, and relevance in a filesystem environment.

ARC[18] used a distance-2 HITS to compile authority lists comparable to human-powered taxonomies. A key idea was inspecting the small text-window surrounding anchors to judge the weight of a link, but this does not map to a filesystem. However, we do consider paths longer than HITS's distance-1.

[7,8] suggested pure connectivity analysis adjustments to improve HITS' accuracy by over 45%. However, ideas focused on treatment of nodes with skewed in/out-degree ratios, which lack an intuitive interpretation in a filesystem. But [8] defends our use of Lucene's VSM, by empirically arguing that there is little difference between VSM, CDR, TLS and Okapi.

[19] use HITS for companion and co-citation identification of related pages, improving on Netscape's "What's related" service. A key idea was to exploit the order of links on the page, but a filesystem's listing order of paths is determined by the operating system (e.g. ascii sort), and not human judgment.

[20] suggested using paths of arbitrary length to abate rare counter-intuitive results of HITS in weakly-connected graphs. We also consider arbitrary path lengths, but our strongly-connected graph has weighted path-lengths which we construct without their expensive matrix exponentiation method.

Using feedback to bias weights [21,20] merits further investigation for a filesystem, where more recently (or frequently) accessed (or modified) nodes could be given artificial boosts, since they appear to interest the user.

The methodology of our work is similar to [22,12], both of which modify PageRank and apply it to a new domain (labeled graph databases, and hyperlinked XML; their semantic links came from schemas and idrefs/xlinks). Our work differs in its applied environment, algorithm and interpretations.

6 Conclusion

We have adopted and adapted a connectivity analysis approach to improve ranked lists in local filesystem search. In a DS context, we hypothesized that favoring relevant files in relevant directories could yield better results.

We provided anecdotal evidence that this may be the case, using a real-world hierarchical corpus with real-world queries. Under MRR, we recognized Lucene as having better ranked results than the other commercial tools. Yet, our two prototypes—running on top of Lucene and a modified-Lucene—improved the MRR of their respective backend by 23% and 36%. Under a per-query normalized metric that shared scores according to relative performance, our prototypes improved the Mean normalized RR of their backends by 19% and 46%.

We conclude that traditional content analysis combined with some connectivity analysis is useful for relevance-ranked results in Desktop Search.

References

1. Dumais, S., Cutrell, E., Cadiz, J., Jancke, G., Sarin, R., Robbins, D.: Stuff I've Seen: a system for personal information retrieval and re-use. In: SIGIR. (2003)
2. Chakrabarti, S., Dom, B., Kumar, S.R., Raghavan, P., Rajagopalan, S., Tomkins, A.: Experiments in topic distillation. SIGIR workshop on Hypertext IR (1998)
3. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank Citation Ranking: Bringing Order to the Web. Stanford Digital Library Technologies Project (1998)
4. Brin, S., Page, L.: The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems **30**(1–7) (1998)
5. Kleinberg, J.: Authoritative sources in a hyperlinked environment. JACM **46**(5) (1999)
6. Marendy, P.: A review of world wide web searching techniques focusing on HITS and related algorithms that utilize the link topology of the world wide web to provide the basis for a structure based search technology (2001)
7. Bharat, K., Henzinger, M.R.: Improved algorithms for topic distillation in a hyperlinked environment. In: SIGIR. (1998)
8. Li, L., Shang, Y., Zhang, W.: Improvement of HITS-based algorithms on web documents. In: WWW. (2002)
9. Ravasio, P., Schär, S.G., Krueger, H.: In Pursuit of Desktop Evolution: User Problems and Practices With Modern Desktop Systems. TOCHI **11**(2) (2004)
10. The Apache Lucene Project: (2006) http://lucene.apache.org.
11. Marchiori, M.: The quest for correct information on the Web: Hyper search engines. Computer Networks and ISDN Systems **29**(8–13) (1997)
12. Guo, L., Shao, F., Botev, C., Shanmugasundaram, J.: XRANK: Ranked keyword search over XML documents. In: SIGMOD. (2003)
13. Noda, T., Helwig, S.: Benchmark Study of Desktop Search Tools. University of Wisconsin-Madison E-Business Consortium (2005)
14. The Java Tutorial: (1995–2005) http://java.sun.com/docs/books/tutorial.
15. Harold, E.: The comp.lang.java FAQ List (1995–1997) http://www.ibiblio.org/java/javafaq.html.
16. Anick, P.: Adapting a full-text information retrieval system to the computer troubleshooting domain. In: SIGIR. (1994)

17. Jansen, B., Spink, A., Bateman, J., Saracevic, T.: Real life information retrieval: a study of user queries on the Web. SIGIR Forum (1998)
18. Chakrabarti, S., Dom, B., Gibson, D., Kleinberg, J., Raghavan, P., Rajagopalan, S.: Automatic resource list compilation by analyzing hyperlink structure and associated text. In: WWW. (1998)
19. Dean, J., Henzinger, M.R.: Finding related pages in the world wide web. Computer Networks **31**(11–16) (1999)
20. Miller, J., Rae, G., Schaefer, F., Ward, L., LoFaro, T., Farahat, A.: Modifications of Kleinberg's HITS algorithm using matrix exponentiation and web log records. In: SIGIR. (2001)
21. Chang, H., Cohn, D., McCallum, A.: Creating Customized Authority Lists. In: ICML. (2000)
22. Balmin, A., Hristidis, V., Papakonstantinou, Y.: ObjectRank: Authority-based keyword search in databases. In: VLDB. (2004)