Community Search Assistant

Natalie S. Glance

Xerox Research Centre Europe
6 chemin de Maupertuis
38240 Meylan, France
glance@xrce.xerox.com

Abstract

This paper describes a new software agent, the community search assistant, which recommends related searches to users of search engines. The community search assistant enables communities of users to search in a collaborative fashion. All queries submitted by the community are stored in the form of a graph. Links are made between queries that are found to be related. Users can peruse the network of related queries in an ordered way: following a path from a first cousin, to a second cousin to a third cousin, etc. to a set of search results. The first key idea behind the use of query graphs is that the determination of relatedness depends on the documents returned by the queries, not on the actual terms in the queries themselves. The second key idea is that the construction of the query graph transforms single user usage of information networks (e.g. search) into collaborative usage: all users can tap into the knowledge base of queries submitted by others.

Introduction

There are three main problems for users of Internet search engines: (1) properly specifying their information need in the form of a query; (2) finding items relevant to their information need, as expressed by the query; and (3) judging the quality of relevant items returned by the search engine.

There has been a great deal of progress in the domain of information retrieval towards addressing the latter two problems. Internet search engines do a fairly good job at discovering items relevant to a query, using primarily statistical techniques to match WWW pages to queries. More recent work has taken advantage of collaborative user behavior, user ratings and/or document hyperlinks to deduce the quality of a WWW page deemed relevant to a given user's query (e.g., Delgaldo, Ishii and Ura 1998; Glance, Arregui, Dardenne 1999; Kleinberg 1998; Page 1997) .

The next step is to address users' fundamental problem: how to help them specify their information need accurately and in such a way that the search engine can best answer their need. A recent analysis of online searches in a library setting has found that almost 50% of all failed searches are caused by semantic errors (Nordlie 1999). The semantic errors occurred either because the user failed to use the appropriate terminology or because the user failed to choose terms at the appropriate level of specificity for his/her information need.

However, in an on-line environment, there is a high probability that some person has formulated a good query representation for any given information need in the recent past. Traces of the search engine Excite[1] reveal that 20-30% of the queries are submitted more than once (Markatos 1999), not taking into account syntactic variants of the same query (e.g., upper/lower case, misspellings, word permutations). This suggests that the collective knowledge of online users, embodied as a set of search queries, can be tapped to help individual users articulate their information needs using the appropriate terminology.

As a concrete example, suppose that a user named Joanna is searching for the WWW page of a conference on mobile computing whose name she does not recall. She is a novice in the area, and does not know whom to ask for help. She types in *mobile computing conference* in her preferred search engine and gets hundreds of responses. Nothing in the top 10 or even top 50 rings a bell, and she does not know how to modify her query. Generalizing her search to *mobile computing* does not help and neither do any of the related searches suggested, for example, by AltaVista[2] (such as *wireless internet mobile computing* and *philips mobile computing*).

Someone else searching for the conference's web site might well remember its name (HUC'99, the International Symposium on Handheld and Ubiquitous Computing). Joanna is unlikely to find it using her query, since the desired result is actually a symposium, not a conference and the keywords are *handheld computing* not *mobile computing*. But how can Joanna take advantage of the fact that someone else has entered the query appropriate to her information need, given that the terms in the queries have zero overlap (*mobile computing* vs. *HUC 1999*)?

Although the two queries have no terms in common, a relationship between the two can be induced from the documents returned to them. For example, a criterion for relatedness could be that of the top ten documents returned

AAAI-2000 Workshop on AI for Web Search

[1] http://www.excite.com/
[2] http://www.altavista.com/

for each query, the two lists have at least one document in common. It turns out that there is a path between these two particular queries that consists of a sequence of related queries. In the first section below, we will describe how to construct a graph of all queries submitted to a search engine within a given period of time. The first key idea is that the determination of relatedness depends on the documents returned by the queries, not on the actual terms in the queries themselves**.**

The second key idea is that the construction of the query graph transforms single user usage of the WWW (e.g. search) into collaborative usage: all users can tap into the knowledge base of queries submitted by others. The community search assistant provides the interface between the query graph and the community of search users. On the one hand, the search assistant allows the user to peruse the network of related queries in an ordered way: following a path from a first cousin, to a second cousin to a third cousin, etc. to a set of results**.** On the other hand, the search assistant records each new search query and updates the query graph incrementally after each new search.

Going back to the Joanna's information need: if she were to use the community search assistant, she might find that *handheld computing conference* is a first cousin to *mobile computing*, among numerous others. (The exact related queries returned will depend on what other users have actually submitted.) Using Google[1] as the reference search engine, these two queries both return "Mobile and Context-Aware Computing at UKC" in their top ten WWW pages, and thus are connected to each other in the graph (as of July 1999).

If the term *handheld* rings a bell, Joanna might well follow the graph in this direction. Next, she might be presented with *handheld computing* as a first cousin to *handheld computing conference* (thus, a second cousin to her original query) because both queries have in common the WWW page "Press Release - BidCom Brings Handheld Computing to The Building Industry." The Web page Joanna is looking for is returned in the top ten documents for this query. Alternatively, she might more rapidly find it by perusing the next set of related queries (now third order cousins to her original query), if some other user has been kind enough to make the query *HUC 1999*.

The rest of the paper is organized as follows. First, we present the approach for constructing the query graph. Next, we describe the community search assistant and provide details of its implementation. Then, we present results obtained from analyzing two different query graphs, both constructed using the same initial set of queries, but using two different search engines. These results indicate that the query graph properties are highly dependent on the search engine used and also highlight the different characteristics of the two search engines employed. Finally, the paper closes with a section on related work and a discussion of the broader utility of the community search assistant.

[1] http://www.google.com/

Query Graphs: Construction

Building an initial query graph requires access to a set of queries made to a search engine or a set of search engines over a period of time. The set of queries might come from a user base as broad as that of the WWW or instead might be limited to the user base within a company Intranet. In principle, the advantage of broader user base is an increased probability of overlap in information needs, for any given information need. The advantage of a more restricted user base is potentially tighter focus and less noise. For the purpose of demonstrating the community assistant, we used a set of 55,000 unique queries obtained over a 12 hour period of August 5, 1999 using MetaSpy[2]. MetaSpy provides a real-time list of all queries to MetaCrawler[3], filtered to remove queries including sexual terms. We kept queries that had between 2 and 9 terms, truncated to the first 100 characters.

Given the set of queries, we next constructed the initial query graph with each node being a query. Two nodes are connected whenever the queries were calculated to be related. One measure of relatedness, for example, is the number of items in common in the top *N* that are returned by a reference search engine. (Note that the reference search engine used to measure relatedness need not be the same search engine as the one from which the set of user queries was obtained.)

In building the query graph from the set of queries, we defined two queries to be related whenever the two queries returned at least one URL in common in the top 10. We also kept track of the number of URLs in common for any

Start with a (possibly) filtered set of queries *Q* (e.g. obtained using Metaspy).

(A) For each unique query *q* in *Q*, build & save list of returned documents:

- Submit *q* to search engine
- Retrieve top *N* responses, save in list *L(q)* (e.g. as URLs)
- Save *q*, *L(q)*, *date* in database

(B) For each query *q* in *Q*, find & save list *R(q)* of related queries:

- Find all queries *q´* in *Q* such that the intersection, *I(q, q´)*, of *L´(q´)* and *L(q)* is not empty
- For all such queries *q´*, add { *q´*, |*I(q, q´)*| } to *R(q)*
- Save *q*, *R(q)*, *date* in database

Figure 1 Query graph construction algorithm

[2] http://www.metaspy.com/spymagic/Spy/

[3] http://www.metacrawler/

two related queries. Fig. 1 outlines the algorithm used for building the graph. The implementation was done using shell scripts and gawk, with a MySQL[1] database serving as the backend for saving the query graph data. We built two versions of the query graph, one using Google to determine relatedness and one using AltaVista. In a later section analyzing our results, we will describe the differences in the two query graphs obtained.

Here are some examples of related searches found in a set of queries submitted to MetaCrawler:

- "Statue of Liberty" and "Ellis Island"
- "Quantum physics" and "copenhagen interpretation"
- "to kill a mockingbird" and "harper lee"
- "frog eye salad" and "Columbian recipes"
- "martin Luther King" and "March on Washington, 1963"
- "Pyrenean Shepherd" and "Berger des Pyrenees"
- "tourism in Spain" and "Santa Marta"

Alternatively, a different measure for determining relatedness can be used when constructing the query graph. For example, two queries can be defined as related when at least one pair of URLs *overlaps* (instead of matching exactly), where the overlap exceeds some threshold. A second possibility is to match queries using other meta-attributes of responses, such as title, summary, retrieved content of document, summary of content of document, etc. In addition, the relative rank of matching items could be taken into account when determining the strength of relatedness between two queries.

Community-Assisted Search

The community search assistant is an agent that maintains the query graph and can be associated with any kind of search engine that uses free text entry. The agent works in parallel with the search engine itself and returns a list of related searches as HTML code, which the search engine can include in addition to its traditional results list.

A screenshot illustrating how the community search assistant works in conjunction with the AltaVista (AV) search engine is shown in Figure 2 (for the purpose of demonstrating the technology, we extracted the basic search functionality of AV from its portal site). From the user's perspective, the search engine returns a list of related queries in addition to the list of items matching his/her query. The related queries are clickable links, causing the new query to be submitted in turn to the community search assistant.

In the background, when a user enters a query, two HTTP requests are sent in parallel: one to the host search engine (in this case, AV) and one to the reference search engine (in this case, Google). Both search engines return a list of items matching the query. The list of items returned by AV is presented to the user; the list of items returned by the

[1] http://www.tcx.se/

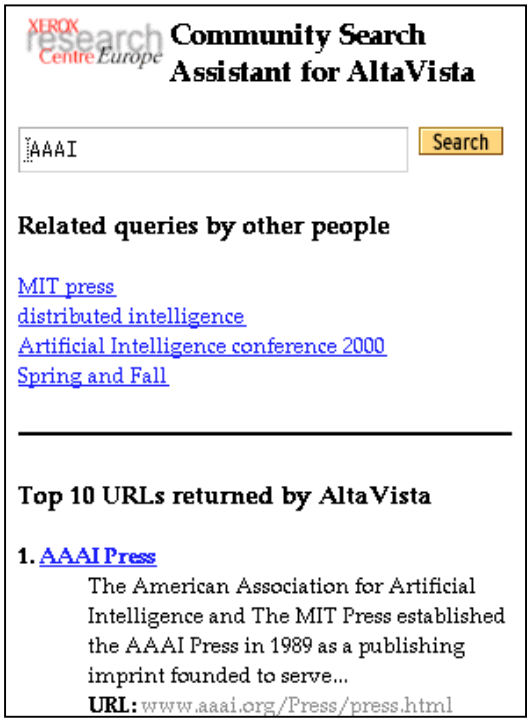

Figure 2 Screenshot of the search assistant

reference search engine is used to calculate the list of related queries and to incrementally update the query graph after each new user query.

Figure 3 shows the algorithm for processing a user query: (1) incrementally updating the query graph; and (2) presenting results to the user.

There are alternative ways to maintain the query graph. For example, the update process might be invoked only for new queries in order to speed up response time. In this case, the links for the query graph could be periodically recalculated off-line, to prevent staleness and to best reflect the current content on the network and its current usage (in the form of queries). In practice, on-the-fly calculation of the set of related queries for a given query is a cheap

Submit q to reference search engine

- Retrieve top N responses and save as list $L(q)$
- Find all queries q' saved in database such that $I(q, q')$ is not empty
- For all queries q', add $\{ q', |I(q, q')| \}$ to $R(q)$
- Save q, $R(q)$, *date* in database

Presenting results to user:

- (Optional) Filter out queries that contain certain keywords
- Return all or first M q' to the user as related searches, ordered by some heuristic (e.g., alphabetical, strength of relatedness)
- If user selects q' in set of related searches, repeat

Figure 3 Processing a user query

calculation, on the order of 1 second for a query graph of size 50,000 queries. The calculation is less than linear in time, thanks to the use of indices in the database.

The related searches presented to the user are ordered by degree of relatedness (number of items in common), highest relatedness first. Only the top 12 most related items are presented. After clicking on a related search, the user is then presented with a set of relevant documents for the related search, as well as a second set of related searches (two hops away from the original query in the graph). In this way, by clicking on a sequence of related searches, the user can follow a path through the query graph. Currently, the user can backtrack by using the "Back" button of the browser or by clicking on the previous query, necessarily in the related query list by symmetry.

The user interface to the search engine could be augmented to provide a more convivial graphical user interface to the query network, using, for example, a tree representation, with the original query at its root. The tree representation can be presented using a conventional tree viewer, or using a more sophisticated visualization such as the hyperbolic browser[1] (Lamping and Rao 1996). Alternatively, the user interface could implement a graph viewer.

Query Graphs: Analysis

We constructed two query graphs from the same initial set of queries, once using Google as the reference search engine, once using AltaVista (AV). The two resulting query graphs had very different characteristics, as summarized in Table 1. First of all, the percentage of queries having at least one neighbor (related query) was much larger using Google: 66% vs. 38%. In addition, the Google-generated graph had 3000 times as many links, which translates into about 85 neighbors per query node (ignoring isolates), on average, vs. 2.8 neighbors per query node for the AV graph.

Table 1 Query graph comparison: Google vs. AltaVista

Reference search engine	**Google**	**AltaVista**
# queries	47,276	54,354
# queries in graph	31,314 (66%)	20,868 (38%)
# isolated queries	15,962 (34%)	33,486 (62%)
# links	1,075,617,393	320,880
# neighbors/query (average)	85 related queries per node	2.8 related queries per node
# neighbors/query (median)	4 related queries per node	1 related query per node
max # neighbors	2,187	65
cluster coefficient	0.72	0.58

[1] http://www.inxight.com/Products/Developer/AD_HT.html

The median number of neighbors per node was 4 per node for the Google graph and 1 per node for AV graph. The clustering coefficient of the Google graph is also higher, 0.72 vs. 0.58. The clustering coefficient indicates to what extent nodes connected to a common node are also connected to each other. A perfectly clustered graph has a clustering coefficient of 1.

Figures 4 and 5 show the probability density for the number of neighbors per node for the Google graph and AV graph, respectively. Both distributions display long tails: a log-log plot of the Google distribution is linear, while the log-log plot for the AV distribution itself has a tail. As indicated in Fig. 4, there exist query nodes with over 1000 neighbors in the Google query graph (up to 2187), while the AV graph has nodes with only 60 or so (up to 65).

At first sight, the qualitative disparity between the two query graphs is surprising. However, we hypothesize that the difference in methodologies employed by the two search engine accounts for the disparity.

Google employs an algorithm that calculates the importance of a page as a function of the web of hyperlinks between pages. Google uses a circular algorithm to

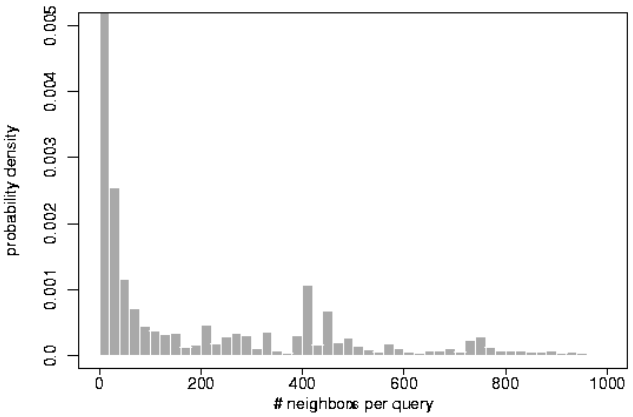

Figure 4 Probability density of # neighbors per query for the Google query graph

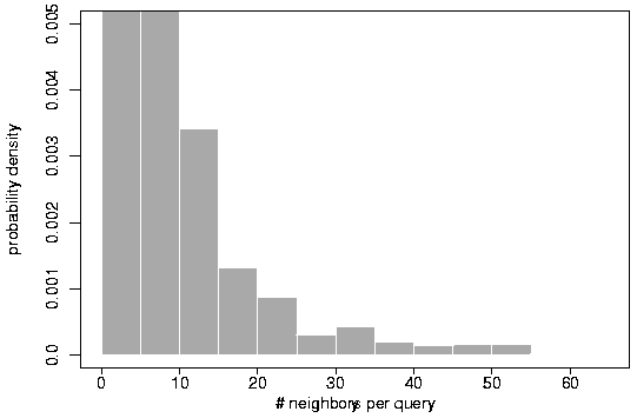

Figure 5 Probability density of # neighbors per query for the AltaVista query graph

calculate importance: a page has high importance if many pages with high importance point to it (Page 1997). Thus, for any given query, hub sites (many out-links) and authoritative sites (many in-links) matching the query will be ranked higher. For its part, AltaVista indexes web pages and uses an inverted index to retrieve URLs that match. Pages that match more of the keywords more often are ranked higher. As a result, Google is more likely than AV to return the same URLs to semantically related queries; Google's results are dependent not only on the keywords in the query, but on the "importance" of the returned URLs. AV, on the other hand, is very good at distinguishing between closely related queries and returns results quite specific to each.

Generalizing from these results, we hypothesize that any search engine that takes into account the popularity or importance of a site/page will likely yield a rich and densely linked query graph.

Also of interest is to understand why the Google graph contains such large clusters. Sifting through the queries that have hundreds or more neighbors, we find that many of these clusters contain queries with many everyday words and no distinguishing key words. For example, many of the queries are song titles (e.g., "that thing you do"; "just the two of us"). Such queries return many of the most densely linked-to URLs (e.g., Microsoft, Netscape, Yahoo) which presumably also happen to contain the words in the query.

One would assume that the community search assistant would perform badly for these kinds of queries when using Google as the reference search engine. In fact, it works much better than one would expect. The community search assistant uses a simple heuristic to rank related queries: it orders related queries by the count of URLs in common. This technique works surprisingly well in finding queries that are semantically related. For example, about half of the top 12 related queries for the two song title queries above are also song titles. In contrast, only one of the first 50 alphabetically ordered related searches is a song title. The filtering mechanism works even though the additional URLs in common typically have nothing to do with music.

Interestingly enough, the AV graph also contains spurious clusters, in which the relations between queries appear to be meaningless. As is well-known, webmasters will at times add invisible text to their pages containing long lists of keywords. As a result, search engines that operate using inverted indices retrieve these pages more frequently. The AV query graph contains large clusters all pointing to a page apparently completely unrelated to the queries themselves, except through this invisible component. The listing of such pages appears to be ephemeral: a reconstruction of the query graph a week later did not contain the same spurious clusters because AV no longer returned the URL for those queries in the cluster. Either AV has mechanisms to recognize such ploys or the pages themselves have short lifecycles. Either way, new spurious clusters appeared to replace the old ones that had disappeared.

Note that in explaining the disparity in the two graphs, we must also differentiate between findings that are likely to hold as the number of queries used to construct the graph increases and findings that will not hold. We have not done tests with larger sets, but results based on subsets of the queries shed light on the stability of these results.

For example, the percentage of isolates is certainly dependent on the number of queries used to construct the graph. Working from a subset, we find a much greater percentage of isolates. Likewise, the density of links remains much greater for Google subgraphs than for AV subgraphs.

Related Work

Recent work in the information retrieval community has addressed collaborative search. One approach has been to use similar past queries to automatically expand new queries, a kind of second order relevance feedback (i.e., documents that correspond well to similar queries provide feedback on the original query as well) (Fitzpatrick and Dent 1997; Brauen 71; Raghavan and Sever 1995). The measure of similarity between queries is a function of the overlap in documents returned by the queries. In all cases, the documents are analyzed linguistically to produce term-frequency vectors, which are then combined with query term-frequency vectors to improve the search process. These augmentation procedures are very costly and generally increase the cost of a search greatly. As a result, these methods are viewed critically by on-line search systems (Fitzpatrick and Dent 1997).

In a similar vein, (Nichols, Twidale and Paice 1997) propose to complement search results with data items from other users' search sessions that are similar to the session in progress. (Glance et. al. 1999) propose that search results be re-ranked against a community profile, which is itself incrementally constructed on the basis of relevance feedback provided by its members over time.

Other research work has focused on collaborative judgments on the quality of WWW pages. For example, Jon Kleinberg has used a graph theoretic approach to identify authorative pages vs. hub pages (Kleinberg 1997). Similarly, Lada Adamic has discovered that the WWW is a small world and suggests using the properties of small world graphs to identify high quality WWW pages as well as hub pages (Adamic 1999). Both approaches mine the web of hyperlinks, as does Google's PageRank algorithm. Recommender systems have also emerged as a way to allow communities of people to collaboratively evaluate the quality of WWW pages and other documents (Delgaldo, Ishii and Ura 1998; Glance, Arregui and Dardenne 1999).

Search engines are beginning to incorporate simple techniques for collaborative search. DirectHit[1] has built a *popularity engine*, which operates using a very simple voting mechanism. Search engines that employ this

[1] http://www.directhit.com/

popularity engine simply track the queries input by users and the links that the users follow. Users vote with their mice: in the future, the same query will yield results whose ordering takes into account previous users' actions. Thus entering a query into a search engine that employs DirectHit's popularity engines will return the most popular results for that query. DirectHit also has a related search technology that works by either broadening or narrowing the user's query (using a subset or superset, respectively, of the user's query terms).

SearchEngineWatch describes the related search features of several major search engines (Sullivan 2000). For example, AltaVista's related search feature works by narrowing the user's query, using query expansion on the original query. For the example query *mobile computing*, AltaVista recommends related searches such as *wireless internet mobile computing* and *philips mobile computing*. Excite suggests key words to add to the original query. The key words are terms appearing commonly in documents returned to the original query.

AskJeeves[1] is a search engine that allows users to enter queries in the form of natural language questions. In the process of answering user queries, the search engine matches user questions to one or more template questions. However, the technology used to do the matching appears to be a mixture of linguistic techniques, e.g., parsing. Also of interest is that AskJeeves also allows people to "peek" at other people's queries.

Discussion

In this paper, we have introduced the collaborative search assistant, a software agent that collects queries from users, incrementally constructs a query graph and interactively recommends related queries. The two main contributions are: (1) the search assistant enables a form of collaborative search by allowing users to tap into the collective history of search queries; and (2) using a simple but effective heuristic to identify related searches, the assistant can recommend related queries that tend to be contextually similar without performing costly linguistic analysis.

A secondary result presented in the paper reveals that the effectiveness of the search assistant depends strongly on the reference search engine used to construct the query graph. In particular, our experiments show that the query graph constructed using Google is orders of magnitudes more densely linked than the one constructed using AltaVista.

The collaborative search assistant can be used to augment any kind of search engine. It is perhaps particularly useful as an enhancement to a knowledge portal, providing a kind of community search and community query memory. In addition, its use is not limited only to search over networked documents, but can also augment, for example, question-answer services. In this case, the overlap in the content of respective answers can be used to identify related questions. Thus, this assistant can also be used to augment "hot-line"/CRM (Customer Relationship Management) applications on the WWW.

[1] http://www.ask.com/

Acknowledgements

Much thanks to Gregory Grefenstette for implementing an initial set of scripts to find related searches among a set of queries and for pointing me to Metaspy.

References

Adamic, L. 1999. Small World Web. In *Proceedings ECDL'99*, Paris, France.

Brauen, T. L. 1971. Document Vector Modification. In *The Smart Retrieval System: Experiments in Automatic Document Processing*, Gerald Salton, Ed., Englewood Cliffs, NJ.

Delgaldo, J., Ishii, N. and Ura, T. 1998. Content-Based Collaborative Information Filtering: Actively Learning to Classify and Recommend Documents. In *Cooperative Information Agents II. Learning, Mobility and Electronic Commerce for Information Discovery on the Internet*. Eds. M. Klusch, G. Weiß, Springer-Verlag, Lecture Notes in Artificial Intelligence Series No. 1435.

Fitzpatrick, Larry and Dent, Mei 1997. Automatic Feedback Using Past Queries: Social Searching? In *Proceedings of SIGIR'97*, Philadelphia, PA.

Glance, N., Arregui, D. and Dardenne, D. 1999. Making recommender systems work for organizations. In *Proceedings of PAAM'99*, London, England.

Glance, N., Grasso, A., Borghoff, U., Snowdon, D. and Willamowski, J. 1999. In *Proceedings HCI'99*, Munich, Germany.

Kleinberg, J. M. 1998 Authorative Sources in a Hyperlinked Environment. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*, Ed. H. Karloff.

Lamping, J. and Rao, R. 1996. The Hyperbolic Browser: A Focus + Context Technique for Visualizing Large Hierarchies. *Journal of Visual Languages and Computing*, 7:33-55.

Markatos, E. 1999. On Caching Search Engine Results. ICS FORTH Technical Report 241.

Nordlie, R. 1999. "User Revealment" – A Comparison of Initial Queries and Ensuing Question Development in Online Searching and in Human Reference Interaction. *In Proceedings of SIGIR99*, Berkeley, CA.

Page, L. 1997. PageRank – Bringing Order to the Web. Stanford Digital Libraries Working Paper 1997-0072.

Raghavan, V.V. and Sever, H. 1995. On the Reuse of Past Optimal Queries. In *Proceedings of SIGIR95*, Seattle, WA.

Sullivan, D. 2000. Search Assistance Features: Related Searches. http://www.searchenginewatch.com/facts/assistance.html#related.