Agglomerative clustering of a search engine query log

Doug Beeferman
Lycos Inc.
400-2 Totten Pond Road
Waltham, MA 02451
dbeeferman@lycos-inc.com

Adam Berger
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
aberger@cs.cmu.edu

ABSTRACT

This paper introduces a technique for mining a collection of user transactions with an Internet search engine to discover clusters of similar queries and similar URLs. The information we exploit is "clickthrough data": each record consists of a user's query to a search engine along with the URL which the user selected from among the candidates offered by the search engine. By viewing this dataset as a bipartite graph, with the vertices on one side corresponding to queries and on the other side to URLs, one can apply an agglomerative clustering algorithm to the graph's vertices to identify related queries and URLs. One noteworthy feature of the proposed algorithm is that it is "content-ignorant"—the algorithm makes no use of the actual *content* of the queries or URLs, but only how they co-occur within the clickthrough data. We describe how to enlist the discovered clusters to assist users in web search, and measure the effectiveness of the discovered clusters in the Lycos search engine.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Clustering; H.3.5 [**Online Information Services**]: Web-based services

1. INTRODUCTION

With the increasing size and popularity of the Internet—there exist over a billion static web pages, and some commercial search engines service tens of millions of queries per day [14]—has evolved an acute need for automatic methods to organize this data. One strategy for bringing a degree of order to a massive, unstructured dataset is to group similar items together. This paper introduces a technique for finding clusters of related queries and related URLs from a collection of user transactions with an Internet search engine.

Most standard document clustering algorithms represent a document as a weighted attribute vector in a high-dimensional space, and group documents based on their proximity according to some distance metric in this space [15, 18]. A distinguishing characteristic of the algorithm proposed here is that it is able to discover highly coherent clusters of URLs without having to embed the pages in a vector space. In fact, the algorithm does not rely at all on the content of the pages, but instead uses co-occurrence information across multiple transactions to guide its clustering.

In proposing to mine information from a large database of transactions, this paper bears resemblance to recent work in market basket analysis, whose goal is to find correlations among individual purchases in a retail setting [1]. That work has subsequently been extended to handle "generalized market baskets," such as documents, which happen to contain words rather than groceries [3].

The clustering strategy described here follows from two related observations. First, the fact that users with the same information need may phrase their query differently to a search engine—`cheetahs` and `wild cats`—but select the same URL from among those offered to them to fulfill that need suggests that the queries are related. Second, the fact that after issuing the same query, users may visit two different URLs—`www.fundz.com` and `www.mutualfundsite.com`, say—is evidence that the URLs are similar.

This paper will proceed as follows. Section 2 surveys prior work in the areas of clustering and query log analysis. Section 3 explains the nature of the "clickthrough data" we use for clustering. Section 4 explains how one can reduce the query/URL clustering algorithm to a problem of partitioning vertices in a bipartite graph. Section 5 illustrates the behavior of the clustering algorithm with statistics and examples of discovered clusters. Section 6 proposes a practical, commercially-inspired method for measuring the quality of different query clustering, and reports how well the algorithms of Section 4 fared in this regard.

2. PREVIOUS WORK

This paper brings together three more or less independent lines of investigation in Internet datamining: query log analysis, web page clustering, and query clustering.

2.1 Query log analysis

Because of the proprietary nature of the information, the scientific literature contain very little analysis of data collected from large-scale commercial search engines. An exception is Silverstein *et. al* [17], who reported statistics accumulated

from a billion-entry set of user queries to Altavista. Earlier, Jansen *et. al* [9] examined a considerably smaller 51,000-query log.

One of the central aims of the present work was to develop a rapid turnaround clustering module capable of identifying and adapting to late-breaking or ephemeral trends in web usage. Therefore, unlike Silverstein's analysis, which involved a query log accumulated over six weeks, we are more interested in discovering information from a single day's records. After having processed the day's queries, a clusterer could provide "fresh" clusters for deployment in the search engine the following day. This is critical for rapid response to news events. Where Sprint and MCI/Worldcom are two unrelated telecommunication companies one day, they might announce a merger the very next day. As soon as possible after the announced merger, a search engine with a rapid-turnaround clustering module could, in response to the query `Sprint`, propose `www.mci.com` as a possibly relevant URL, and suggest `MCI/Worldcom` as a related query.

2.2 Clustering URLs

A high quality URL clustering algorithm could help in automating the process of ontology generation (the task which Yahoo and OpenDirectory human employees perform), organizing bookmark files into categories, constructing a user-specific profile of "pages this person finds interesting," and grouping the results of a web search by category. Not surprisingly, the past few years have witnessed a great deal of interest in this area [2, 4, 19].

Researchers have been investigating the more general problem of document clustering algorithms for decades, and it makes sense to consider how well these approaches—which do not exploit the correlations between documents and queries inherent in a search engine transaction database—would fare on a subset of the Internet. One popular technique is *k-means* [8], whose running time is linear in the number of documents n but is most effective in the rather restrictive scenario when the documents lie in a spherical configuration with respect to the underlying metric [19]. Also popular are *hierarchical agglomerative clustering* (HAC) techniques, typically at least quadratic in n, which repeatedly find the closest two documents and merge them. Both methods are susceptible to undesirable behavior when faced with outliers.

Our approach falls into the HAC category, but differs from traditional techniques in one significant way. Namely, the distance between two documents can be evaluated without examining the contents of those documents, a property we shall refer to as "content-ignorance." This stands in contrast to traditional "content-aware" clustering algorithms, which typically use as a distance between documents some function of the fraction of tokens they have in common.

Clustering web pages by content may require storing and manipulating a staggeringly large amount of data: by early 2000, there were at least a billion pages on the Internet accessible by commercial search engines [7]. Content-ignorance can obviously be a valuable property when handling a dataset of this scale. It can also be applicable, at least in principle, in settings where content-aware clustering is not, including:

- *Text-free pages*: A distance function calculated from the text of a web page isn't capable of recognizing the correspondence between a web page containing just a picture of an emu and another describing the appearance and behavior of a emu.
- *Pages with restricted-access*: URLs may be password-protected or temporarily unavailable, rendering them unavailable to a clustering procedure which relies on the content of the page.
- *Pages with dynamic content*: A URL might always point to a company's web page, but the contents of that page are likely to change regularly, updated with news about the company. A content-aware clusterer is more susceptible to placing a URL in different clusters as the page is modified.

The last and perhaps most important advantage is that content-ignorant clustering can be implemented relatively more efficiently than standard agglomerative techniques. This will become apparent in Section 4.1, which describes how the graph-based distance function we use allows for efficient memoization as the algorithm proceeds, drastically reducing the computational cost of standard HAC methods.

Another flavor of document clustering, which we mention only to distinguish from the clustering discussed here, is locating groups of identical or nearly-equivalent documents in a large database like the Internet. Broder *et. al* [4] introduced a technique which involves calculating a "fingerprint" of a web page based on contiguous subsequences of tokens in the page, and evaluated the algorithm on 30 million web pages collected by AltaVista in April 1996. Shivakumar and Garcia-Molina [16] also examine syntactic resemblance in the context of detecting copyright violation, and applied their algorithm to a collection of 50,000 netnews articles. It appears that finding equivalence classes of URLs, all of which point to the same (or nearly identical) files, requires a content-aware strategy.

2.3 Clustering queries

Clustering queries submitted to search engines appears to be a rather less explored problem than clustering web pages, though there are practical, commercial applications for a high-quality query clusterer. For instance, if a user submits to a search engine a query q which is a member of a cluster $\mathcal{C}$, the search engine could suggest as alternate, related queries the other members of $\mathcal{C}$. Such a "related query" tool is deployed in the Lycos search engine; the component appears in Figure 1.

A recent, independent survey of 40,000 web users found that after a failed search, 76% of users try rephrasing their query on the same search engine, instead of resorting to a different search engine [14]. Moreover, historical trends of search engine usage patterns show that users have begun to rely more—a twenty percent increase during 1999—on single-term queries. This suggests that many users rely on search engines to help them home in an optimal representation of their information need.

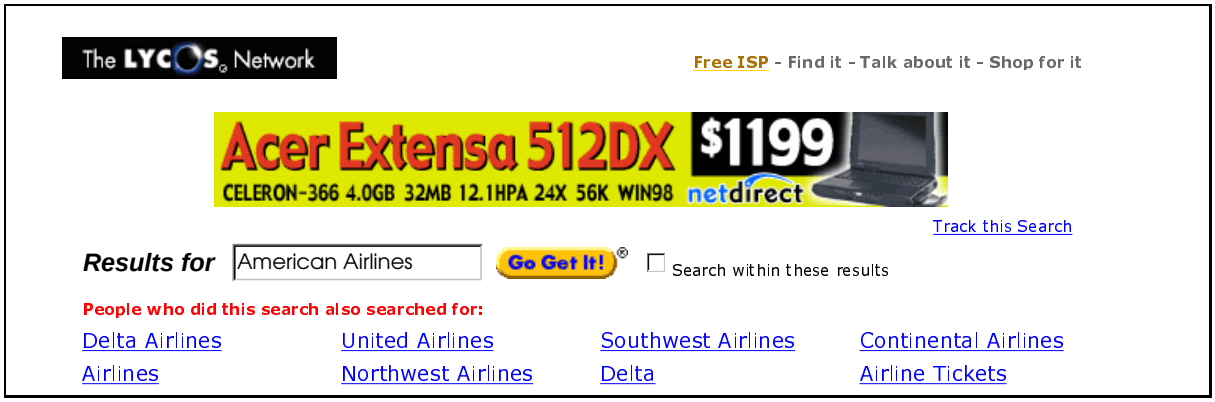

Figure 1: Many commercial search engines offer users the opportunity to rephrase their information need by suggesting alternate queries. Shown is the top of a page generated by Lycos in response to the query `American Airlines`. A query clustering algorithm could provide such a list of suggestions by offering, in response to a query q, the other members of the cluster containing q.

3. CLICKTHROUGH DATA

The `http` protocol allows commercial search engines the ability to record a great deal of information about their users—the name and IP address of the machine which sent the request, the type of web browser running on the machine, the screen resolution of the machine, and so on. Here we are interested only in the sequence of characters comprising the query submitted by the user, and the URL selected by that user from among the choices presented by the search engine. Table 1 lists a small extract of clickthrough records (`query, URL`) from a recent Lycos log.

For the experiments reported in this paper, we applied a simple filter to detect and remove those records containing objectionable content such as pornography and hate speech. This eliminated about 20% of the records gathered, including an undetermined number of false positives and negatives. We also mapped the query characters to lowercase and converted a sequence of one or more spaces to +, but otherwise performed no processing of the queries submitted to Lycos.

We avoid significant preprocessing of the search engine logfile to underscore the "pushbutton" nature of the proposed clustering algorithm. Clearly, however, even a minimal amount of preprocessing could go a long way in helping the clustering. For instance, the query in the second entry of Table 1 contains a comma, which distinguishes it from the semantically identical `missoula+mt`. And the ninth query in the table contains an easily-detectable misspelling, which distinguishes it from `bulldog+homepage`. Query and URL canonicalization aren't really necessary, however, since the algorithm introduced in the next section will uncover these sorts of equivalences anyway.

4. GRAPH-BASED ITERATIVE CLUSTERING

We now describe a technique for using a collection of clickthrough records to simultaneously discover (a) disjoint sets of queries, where the members of a single group represent (roughly speaking) different ways of expressing a similar information need, and (b) disjoint sets of URLs, where the members of a single group represent different web pages corresponding to similar information needs.

Starting from a corpus of clickthrough data, the first step is to construct a bipartite graph where the vertices on one side correspond to unique queries, the vertices on the other side to unique URLs, and edges exist between a query and URL vertex when they co-occurred in a clickthrough record. Algorithm 1 details this construction.

Algorithm 1: *Bipartite graph construction*

Input: Clickthrough data $\mathcal{C}$ in the form of (`query, URL`) pairs, as in Table 1.

Output: Bipartite graph $\mathcal{G}$

1. Collect a set $\mathcal{Q}$ of unique queries from $\mathcal{C}$
2. Collect a set $\mathcal{U}$ of unique urls from $\mathcal{C}$
3. For each of the n unique queries, create a "white" vertex in $\mathcal{G}$.
4. For each of the m unique urls, create a "black" vertex in $\mathcal{G}$.
5. If query q appeared with url u, then place an edge in $\mathcal{G}$ between the corresponding white and black vertices.

Intuitively, if we write $\mathcal{N}(x)$ for the set of vertices neighboring vertex x in a graph, then vertex y is "similar" to vertex x if $\mathcal{N}(x)$ and $\mathcal{N}(y)$ have a large overlap. More formally, define the *similarity* $\sigma(x, y)$ between vertices x and y by

$$\sigma(x,y) \stackrel{\text{def}}{=} \begin{cases} \dfrac{\mathcal{N}(x) \cap \mathcal{N}(y)}{\mathcal{N}(x) \cup \mathcal{N}(y)}, & \text{if } |\mathcal{N}(x) \cup \mathcal{N}(y)| > 0 \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

The similarity between two vertices lies in the range $[0 \ldots 1]$: 0 if the vertices have no neighbors in common, and 1 if they have exactly the same neighbors. This definition of similarity is straightforward, intuitive, and convenient to work with, but does suffer from not distinguishing between two vertices which each have the same neighbor and two vertices which each have the same two neighbors.

felony	jud13.flcourts.org/felony.html
missoula,+mt	missoula.bigsky.net/score/
feeding+infants+solid+foods	members.tripod.com/drlee90/solid.html
colorado+lotto+results	www.co-lotto.com/
northern+blot	www.invitrogen.com/expressions/1196-3.html
wildflowers	www.life.ca/nl/43/flowers.html
ocean+whales	playmaui.com/ocnraftn.html
ralph+lauren+polo	www.shopbiltmore.com/dir/stores/polo.htm
bulldog+hompage	www.adognet.com/breeds/2abulm01.html
lyrics	www.geocities.com/timessquare/cauldron/8071
churches+in+atlanta	acme-atlanta.com/religion/christn.html
retail+employment	www.crabtree-evelyn.com/employ/retail.html
illinois+mortgage+brokers	www.birdview.com/ypages2/c3.htm
stock+exchange+of+singapore	www.ses.com.sg
front+office+software	www.saleslogic.com/saleslogix.phtml
free+3d+home+architect	www.adfoto.com/ads1/homeplans.shtml
country+inns+sale	innmarketing.com/form.html
free+desktop+wallpaper	www.snap-shot.com/photos/fireworks/
automotive+marketing+research	www.barndoors.com/rcmresources.htm
router+basics	www.wheretobuy.com/prdct/706/55.html

Table 1: A small excerpt of Lycos clickthrough records—user query and the selected URL—for a single day in February 2000.

To use $\mathcal{G}$ to discover groups of similar queries, one could simply apply an agglomerative clustering algorithm to the white nodes of $\mathcal{G}$, using $1-\sigma$ as the distance between nodes[1] One could apply the same procedure to black nodes to find groups of related URLs. Algorithm 2 iterates between these two clustering procedures, first combining the two most similar queries, then combining the two most similar URLs, and repeating.

Algorithm 2: *Agglomerative iterative clustering*

Input: Bipartite graph $\mathcal{G}$

Output: A new bipartite graph $\mathcal{G}'$: each white (black) vertex of $\mathcal{G}'$ corresponds to one or more white (black) vertices of $\mathcal{G}$.

1. Score all pairs of white vertices in $\mathcal{G}$ according to (1)
2. Merge the two white vertices w_i, w_j for which $\sigma(w_i, w_j)$ is largest.
3. Score all pairs of black vertices in $\mathcal{G}$ according to (1)
4. Merge the two black vertices b_i, b_j for which $\sigma(b_i, b_j)$ is largest.
5. Unless a termination condition applies, go to step 1.

At first glance, it might not be entirely clear why an iterative approach is necessary. Why not simply cluster the white (query) vertices first, and then cluster the black (URL) vertices? The purpose of iteration is to reveal similarities between vertices which aren't apparent in the original graph. Figure 2 provides a simple example.

[1]That $1-\sigma$ is indeed a metric is not difficult to prove, but lies outside the scope of this paper.

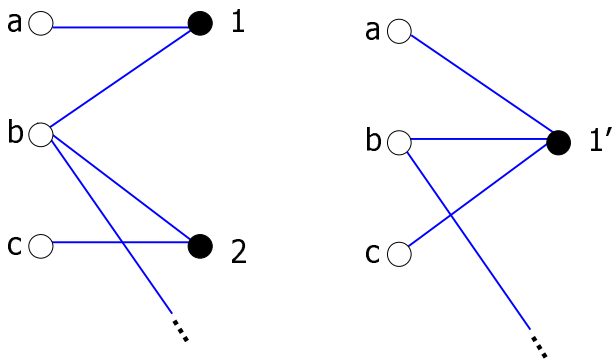

Figure 2: *Left:* White vertices a and c have a similarity $\sigma(a,c)=0$. *Right:* After merging black vertices 1 and 2, which have a similarity of $1/3$, vertices a and c suddenly appear quite similar. The power of agglomerative clustering lies in its ability to uncover such latent relationships between nodes.

Algorithm 2 leaves undefined the matter of a termination condition. One reasonable strategy is to iterate until the resulting graph consists of connected components, each with a single query and URL vertex. In other words, continue clustering until

$$\max_{q_i,q_j\in\mathcal{Q}} \sigma(q_i,q_j)=0 \quad \text{and} \quad \max_{u_i,u_j\in\mathcal{U}} \sigma(u_i,u_j)=0$$

This condition is almost certainly too severe, and we are

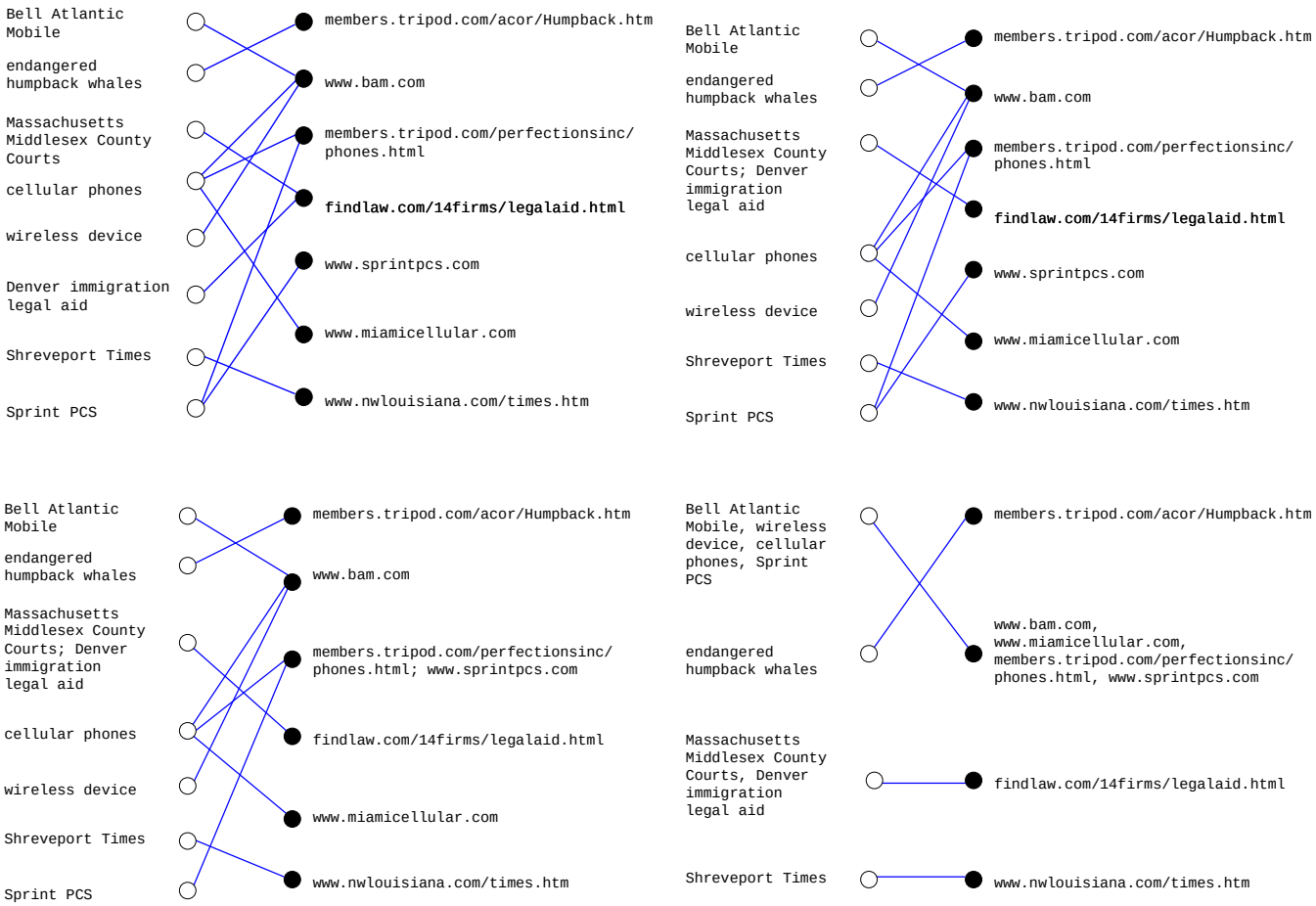

Figure 3: Progression of Algorithm 2 on a small collection of clickthrough records. Clockwise, from top left: (a) The original bipartite graph, unclustered. (b) After the initial query clustering (c) After the initial URL clustering (d) At termination: the edges here correspond to connected components in the original graph.

currently exploring other threshold criteria. Allowing Algorithm 2 to run until the above termination condition (i.e. finding the connected components in $\mathcal{G}$) is a well-studied task for which there exist several viable solutions, including simple depth-first search [5]. The matter of knowing when to stop in hierarchical agglomerative clustering algorithms has been explored, but not resolved [12].

4.1 Complexity of clustering

If $\mathcal{G}$ contains n_w white vertices and n_b black vertices, a naive analysis suggests that Algorithm 2 requires $\Theta(n_w^2 + n_b^2)$ operations per iteration to calculate all pairwise distances and select the best merge. Fortunately, taking note of a few simple observations reduces the complexity of the algorithm dramatically.

We define two vertices to be *siblings* if they share a neighbor. In principle, $\mathcal{G}$ may contain up to $(n_w^2 - n_w)/2$ unique white-vertex siblings. But since bipartite graphs constructed from clickthrough data using Algorithm 1 are (in our experience) quite sparse, the number of siblings is in practice much smaller (essentially constant). Denoting by $|\mathcal{N}|_{\max}$ the maximum number of neighbors of any vertex in $\mathcal{G}$, the number of sibling white vertices in $\mathcal{G}$ before Algorithm 2 commences is bounded above by $n_w |\mathcal{N}|_{\max}^2$. Moreover, it is fairly intuitive that the number of siblings in $\mathcal{G}$ can only decrease during the execution of Algorithm 2.

The key to an efficient implementation of Algorithm 2 is recognizing that after calculating $\sigma(x, y)$ once for all relevant vertex pairs, only a few $\sigma(x, y)$ values will change at any iteration. Specifically, after merging two white nodes w_1 and w_2, the distance between b_1 and b_2 changes only if

$$w_1 \in \mathcal{N}(b_1) \cup \mathcal{N}(b_2) \quad \text{and} \quad w_2 \in \mathcal{N}(b_1) \cup \mathcal{N}(b_2) \tag{2}$$

So after a one-time computation to calculate all σ values, Algorithm 2 requires at most $2|\mathcal{N}|_{\max}$ distance calculations after each merge.

Putting these observations together, we arrive at a conservative upper bound for the number of operations required

in Algorithm 2:

$$\underbrace{(n_w + n_b)|\mathcal{N}|_{\max}^2}_{\textit{One-time computation}} + m\underbrace{(4|\mathcal{N}|_{\max})}_{\textit{per iteration}}$$

where m is the number of iterations (merges) desired.

5. RESULTS

This section describes the result of applying Algorithms 1 and 2 to a collection of clickthrough records provided by Lycos. The two algorithms were run on a 266 MHz processor in a Sun UltraSPARC workstation with 1.5GB of physical memory, and required about ten wall clock hours of computation.

We started with a set of 500,000 clickthrough records, a portion of a single day's transactions on the Lycos search engine in early 2000. Applying Algorithm 1 on this dataset generated a graph $\mathcal{G}$ with 243,595 white nodes and 361,906 black nodes. The graph $\mathcal{G}$ is, not surprisingly, quite sparse: though a fully connected graph would have had $n_w(n_w-1)/2 = 29$ billion white siblings, $\mathcal{G}$ had fewer than two million. Table 5 contains the white and black "sibling density" of $\mathcal{G}$, along with a number of other statistics from this dataset.

Clickthrough records:	500,000
Unique queries:	243,595
Unique URLs:	361,906
Query sibling pairs:	1,895,111
Query edge density:	6.38×10^{-5}
URL sibling pairs:	476,505
URL edge density:	7.27×10^{-6}

Table 2: Statistics of the Lycos clickthrough database used in the experiments reported here. "Query sibling pairs" are queries which both occurred in a clickthrough record with the same URL; in terms of the corresponding bipartite graph, these are pairs of white vertices with a common neighbor. This value limits the number of iterations Algorithm 2 can perform on the dataset before the similarity between any two candidate query vertices becomes zero.

6. USING QUERY CLUSTERS TO ENHANCE WEB SEARCH

As do many commercial web search engines, Lycos offers a list of alternate query formulations for selected queries. The selection of suggestions is driven by an algorithm which relies on the search refinement behavior of past Lycos users. A (query, suggestion) pair is selected for inclusion in this baseline system if the number of past user sessions the pair has co-occurred in meets various information-theoretic criteria.

The baseline algorithm has the weakness that it is data-intensive and based exclusively on having witnessed its specific suggestions in hindsight. This shortcoming means the algorithm is ill-equipped to track emerging topics of interest to users, such as events in the news and newly publicized web sites.

A tempting strategy for assessing the quality of a query suggestor is to inspect the suggestions and see how reasonable they look. But in fact it may be ill-advised to judge query/suggestion pairs on the basis of whether they "make sense together." Some perfectly valid pairs seem unrelated at first glance—the query `WWF`, for example, may seem to be a spurious suggestion for the input query `Pokemon`, but in fact the suggestion makes sense: the demographic profiles of searchers for these items are similar.

The most important feature of a measure of cluster quality is that it be suitable to its intended application. In this spirit, we employ as an objective function the *clickthrough rate* of a given list of suggestions: the proportion of instances the list was presented to the user that one of its elements was clicked on. If the rest of the search experience is held constant, this function is a concrete measure of the value provided to the search engine and to the end user by a particular suggestion method.

We experimented with three different methods for building "suggestion lists" (a list of alternate formulations of a query):

Baseline: the standard Lycos query-suggestion algorithm.

Full-replacement: For a subset of search requests $\hat{\mathcal{Q}} \in \mathcal{Q}$ selected blindly from the list of most frequently accessed search requests at Lycos, replace the default suggestion list with up to eight members of the cluster that contains q. We only apply this technique to queries whose constituent words all appear in some cluster. To decide which members of the cluster should participate in the suggestion list, we select those members which occurred most frequently in the 500,000 clickthrough records from which the clustering was learned.

Hybrid: Replace *some*, but not all of the suggested rephrasings with the clustering suggestions. More specifically, we replace the "weakest" (lowest-ranking) default suggestions with the best available clustering-based suggestion, skipping replacements that would introduce repeats into the suggestion list.

For each method, we measured its impact on the live Lycos search engine by enaging the associated suggestion method for a two-day period. Each period included a Friday and a Saturday in April, 2000. Table 3 shows the aggregate results, which suggest that the hybrid method outperforms the baseline and full-replacement strategies.

Offline, we later measured the clickthrough rates on the suggestions that are triggered by queries in $\hat{\mathcal{Q}}$. Table 4 shows the cumulative results for five blindly-selected popular queries, and the aggregate statistics over all five queries. Table 5 shows the list of suggested replacements for the six blindly selected queries.

The experiments demonstrate that the hybrid method slightly outperforms both the baseline method and the cluster-based method in the aggregate, although it is inferior for certain individual queries.

REUTERS+CUBA ELIAN+GONZALEZ ULRIKA+JOHNSON MYRNA+GOOSSEN GONZALEZ+ELIAN GONZALEZ,+ELIAN ELIAN REUTERS+POLITICS (1371)
FONTS FREE+SCRIPT+FONTS FREE+FONTS (101)
NETWORK+UTILITIES NETWORK+ANALYSER SERVER+MIRRORING NETWORK+SNIFFER WINDOWS+PACKET+SNIFFER PACKET+SNIFFER HTTP+SNIFFER NETWORK+UTILITY SOFTWARE+AND+HARDWARE+AUDIT+TOOLS NETWORK+SECURITY+TOOLS AUDIT+SOFTWARE SOFTWARE+AUDITING AUDITING+SOFTWARE NET+AUDITING NETWORKING+TOOLS (100)
ANOREXIA ANOREXIA+NERVOSA BULIMIA+NERVOSA STARVATION CAUSES+OF+ANOREXIA BULIMIA ANOREXIA,BULIMIA (64)
GUNS RIFLES BERETTA FIREARMS TOY+GUNS (63)
JAVA+APPLETS JAVA FREE+JAVA+APPLETS FREE+CHAT+APPLET FREE+JAVA+BUTTONS FREE+JAVASCRIPT+AND+APPLETS JAVABOUTIQUE APPLETS (49)
ARMOR MEDIEVAL+WEAPONS AX MIDEVAL+ARMOR (20)
CROHNS CROHNS+DISEASE CROHN'S ULCERATIVE+COLLITIS (17)
AUTOMOTIVE+ACCESSORIES HEADLIGHT+COVERS CAR+SPOILERS ONLINE+AUTO+PARTS CAR,ACCESSORIES,LIGHT (11)

Figure 4: Selected query clusters generated automatically from by Algorithm 2 on a collection of $500,000$ clickthrough records provided by Lycos. The algorithm had run for $100,000$ iterations. The numbers in parentheses are the total number of appearances of all members of the cluster within the clickthrough data.

period	strategy	impressions	clicks	clickthrough rate
April 7-8	baseline	6,120,943	71138	1.16%
April 14-15	hybrid	6,058,757	79515	1.31%
April 21-22	full replacement	5,985997	61377	1.03%

Table 3: Aggregated clickthrough results for the three query-suggestion methods described in this section. "Impressions" is the number of times the list of suggestions was presented to a user during this time period.

For established, high-traffic queries such as `American Airlines`, the baseline algorithm outperforms both its competitors, because its large store of highly targeted user session data for this query is well matched with what is of interest to users in the future. But for `bibliofind`, a rarer query of emerging interest to select users, session data is not available in sufficient quantity for the baseline algorithm to identify a reasonable set of alternatives. For such queries the cluster-based suggestion list appears preferable.

7. DISCUSSION

The central contribution of this paper is a substantiated claim that a collection of web search engine transactions is a fertile source from which to harvest clusters of related queries and related web pages. The proposed strategy is to view the clickthrough records as a bipartite graph, and apply an iterative, agglomerative clustering algorithm to the vertices of this graph.

The notion of applying datamining techniques to information discovery in a database of web transactions is not a new one. Mobasher *et. al* suggest applying the technique of association rule discovery to find correlations in web server access logs. For instance, the algorithm might discover that clients who requested file x were subsequently very likely to request file y [13].

In recasting document clustering as a problem in discovering similar vertices in a graph, this work bears some semblance to recent work by Boley *et. al* [2]. Their strategy is to use association rule discovery to find pairs of similar document (using the document contents), building a graph whose vertices correspond to documents and edges correspond to discovered associations, and then using standard graph partitioning techniques to find related vertices.

Though this paper has focused on the web domain, we anticipate that the graph-based technique should work equally well in other settings. For instance, one could apply Algorithms 1 and 2 to a database of online purchase transactions to cluster consumers and items. Grouping together related customers is closely related to the emerging problem of *collaborative filtering*: mining large databases of user transactions in order to recommend items—books, movies, or even politicians—which the system expects the user might find of interest [6, 11].

Within a search engine, page clustering could serve two purposes. First is to organize all web pages *a priori* into groups, independent of any particular user. Second to cluster just those pages suggested to a user in response to their query [19], as an aid in browsing the search results. Although we have conducted this research with an eye to the former setting, one could certainly apply graph-based clustering just to a small set of URLs, by selecting out those clickthrough records containing the URLs, and applying the clustering algorithm to this dataset.

Section 6 described a practical strategy for measuring the quality of a query clustering: one clustering is superior to

www.nationalcar.com/ flyaow.com/carrentals.shtml www.hertz.com/ www.bnm.com/rcar.htm www.nationalcar-europe.com/ (80)
www.ragingbull.com/ www.ragingbull.com hedge-hog.com/bbs/messages/166.html quote.ragingbull.com/ (30)
metallica.andmuchmore.com/ www.www.metallica.com/ www.metallica.com rollingstone.lycos.com/artists/default.asp?LookUpString=227 eonline.com/Reviews/Music/Scoop/000314b.html (27)
www.adopting.com/agencies3.html www.adopt.org/ (13)
www.history-of-rock.com www.rockhall.com/ www.experience.org (12)
www.princesscruises.com/ www.awcv.com/princess.html www.freecruisequote.com www.hotcruise.com/princesscruises.htm (9)
help-wanted.net www.helpwusa.com www.helpwantedpage.com/ (6)

Figure 5: Selected URL clusters generated automatically from by Algorithm 2 on the same collection of clickthrough records as described above.

another if people find the first clustering more useful in interacting with a web search engine. We plan to perform a complementary set of experiments to evaluate quality of the URL clusters, again by observing how the behavior of users interacting with the Lycos search engine changes as they are provided access to clusters calculated from query log data.

Despite the efficiency measures suggested in Section 4.1, Algorithm 2 is still a form of hierarchical agglomerative clustering, and, as presented, only merges one query and document cluster per iteration. We are exploring greedy strategies for speeding up the clustering process by performing many merges at once.

One question addressed but not resolved by this work is how best to combine the complementary strategies of content-ignorant and content-aware clustering. Each method has weaknesses. Clickthrough data suffers in that "unpopular" URLs—those never appearing in the clickthrough data because they were never selected by a user—can't be processed. A content-aware clusterer can examine the content of *all* pages, popular or not, but fails to recognize the potentially rich source of knowledge about queries and URLs latent within a search engine log.

8. REFERENCES

[1] R. Agrawal, T. Imielinski and A. Swami (1993). Mining association rules between sets of items in a large database. *Proceedings of the ACM Conference on Management of Data (SIGMOD'93)*. Washington, DC.

[2] D. Boley, M. Gini, R. Gross, S. Han, K. Hastings,G. Karypis, V. Kumar, B. Mobasher and J. Moore (1999). Partitioning-based clustering for web document categorization. *Journal of Decision Support Systems*.

[3] S. Brin, R. Motwani and C. Silverstein (1997). Beyond market baskets: Generalizing association rules to correlations. *Proceedings of the ACM Conference on Management of Data (SIGMOD'97)*. Tuscon, AZ.

[4] A. Broder, S. Glassman and M. Manasse (1997). Syntactic clustering of the web. *Proceedings of the Sixth International World Wide Web Conference*. Santa Clara, CA.

[5] T. Cormen, C. Leiserson and R. Rivest (1990). *Introduction to Algorithms*. MIT Press.

[6] J. Herlocker, J. Konstan, A. Borchers and J. Riedl (1999). Am algorithmic framework for performing collaborative filtering. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR'99)*. Berkeley, CA.

[7] Inktomi Corporate Web Page (2000). `www.inktomi.com`.

[8] A. Jain and R. Dubes (1988). *Algorithms for clustering data*. Prentice Hall.

[9] B. Jansen, A. Spink, J. Bateman and T. Saracevic (1998). Real-life information retrieval: A study of user queries on the Web. *SIGIR Forum* **32(1)**, 5–17.

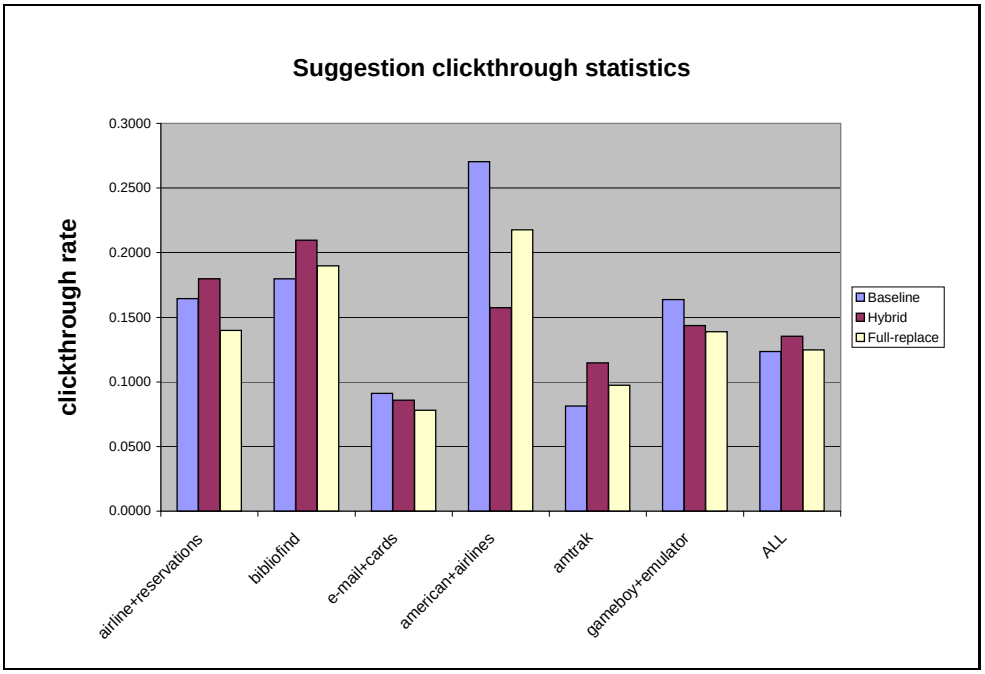

Table 4: Clickthrough rates on a per-query basis for six blindly-selected queries, according to the three different strategues.

[10] J. Kleinberg (1998). Authoritative sources in a hyperlinked environment. *Proc. 9th ACM-SIAM Symposium on Discrete Algorithms.*

[11] J. Konstan, B. Miller, D. Maltz, J. Herlocker, L. Gordon and J. Riedl (1997). GroupLens: Applying collaborative filtering to Usenet news. *Communications of the ACM* **40(3)**: 77–87.

[12] G. Millighan and M. Cooper (1985). An examination of procedures for detecting the number of clusters in a data set. *Psychometrika* **50**, 159–179.

[13] B. Mobasher, N. Jain, E. Han, J. Srivastava (1996). Web Mining: Pattern discovery from Word Wide Web Transactions. *Technical Report TR96-050*, Department of Computer Science, University of Minnesota.

[14] NPD Search and Portal Site Survey (2000). Published by NPD New Media Services. Reported in `www.searchenginewatch.com`.

[15] G. Salton (1989). *Automatic text processing.* Addison-Wesley.

[16] N. Shivakumar and H. Garcia-Molina (1996). Building a scalable and accurate copy detection mechanism. *Proceedings of the First ACM Conference on Digital Libraries (DL'96).* Bethesda, MD.

[17] C. Silverstein, M. Henzinger, H. Marais and M. Moricz (1998). Analysis of a very large AltaVista query log. *DEC SRC Technical Note 1998-014.*

[18] P. Willet (1988). Recent trends in hierarchical document clustering: A critical review. *Information Processing and Management* **24**: 577–597.

[19] O. Zamir and O. Etzioni (1998). Web document clustering: A feasibility demonstration. *Proceedings of the ACM Conference on Information Retrieval (SIGIR'98)*, Melbourne, Australia.

	Baseline	Hybrid	Full-replace
airline+ reservations	Southwest airlines, airline tickets, American Airlines, Continental Airlines, Airline travel	Southwest airlines, airline tickets, American Airlines, Travel agencies, plane tickets	Travel agencies, plane tickets, travel agency, low airfare, hotel reservations, hotels carribean, travelosity, sabre.com
bibliofind	Adebooks, Bookfinder, Abe	Rare books, used books, books used	Rare books, used books, books used, out of print botany books, out-of-print books, out of print books, books antique, rare books'
e-mail+cards	Free e-mail cards	Talk City	Talk City, swan backgrounds, insurance seminar, creatacard, postnet.com, greatingcards, 4anything.com, lariam discussion groups
american+ airlines	Delta Airlines, United Airlines, Southwest Airlines, Continental Airlines, Airlines, Northwest Airlines, Delta, Airline tickets	Delta airlines, United Airlines, Southwest Airlines, Continental Airlines, www.aa.com, aa.com, air fairs, american airline	www.aa.com, aa.com, air fairs, american airline, 'american airlines', american air, amr.com, american eagle airlines
amtrak	Amtrack, Greyhound, Trains, train, Amtrac, Greyhound bus, Southwest airlines, Delta	Amtrack, Greyhound, Trains, Amtrak schedules, www.amtrack.com, amtrack reservations, amtrack.com, train amtrak	Amtrak schedules, Amtrack, www.amtrack.com, amtrack reservations, amtrack.com, train amtrak, amtrack train schedule, train amtrack
gameboy+ emulator	Gameboy ROMs, emulator, Pokemon ROMs	Gameboy ROMs, emulator, Nintendo ROMs	Emulator, Nintendo ROMs, NES ROM

Table 5: Suggested replacements for the six blindly-selected queries.