IEEE 2012 Transactions on Knowledge and Data Engineering, Volume: 24 , Issue: 5

Organizing User Search Histories

Heasoo Hwang, Hady W. Lauw, Lise Getoor and Alexandros Ntoulas

Abstract—Users are increasingly pursuing complex task-oriented goals on the Web, such as making travel arrangements, managing finances or planning purchases. To this end, they usually break down the tasks into a few co-dependent steps and issue multiple queries around these steps repeatedly over long periods of time. To better support users in their long-term information quests on the Web, search engines keep track of their queries and clicks while searching online. In this paper, we study the problem of organizing a user's historical queries into groups in a dynamic and automated fashion. Automatically identifying query groups is helpful for a number of different search engine components and applications, such as query suggestions, result ranking, query alterations, sessionization, and collaborative search. In our approach, we go beyond approaches that rely on textual similarity or time thresholds, and we propose a more robust approach that leverages search query logs. We experimentally study the performance of different techniques, and showcase their potential, especially when combined together.

Index Terms—user history, search history, query clustering, query reformulation, click graph, task identification

1 Introduction

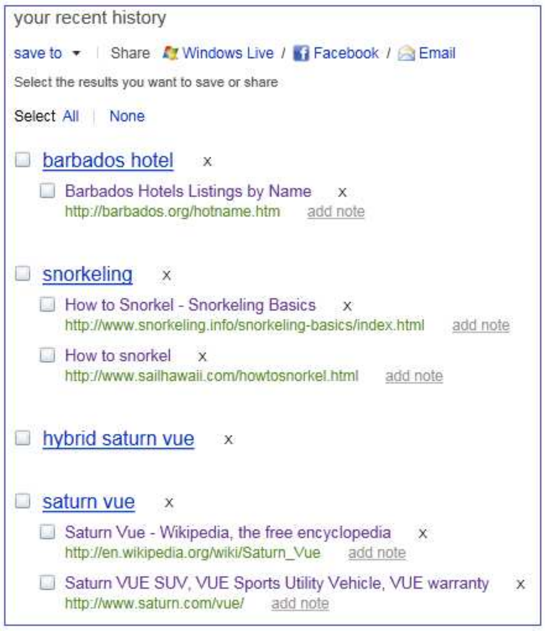
As the size and richness of information on the Web grows, so does the variety and the complexity of tasks that users try to accomplish online. Users are no longer content with issuing simple navigational queries. Various studies on query logs (e.g., Yahoo's [1] and AltaVista's [2]) reveal that only about 20% of queries are navigational. The rest are informational or transactional in nature. This is because users now pursue much broader informational and task-oriented goals such as arranging for future travel, managing their finances, or planning their purchase decisions. However, the primary means of accessing information online is still through keyword queries to a search engine. A complex task such as travel arrangement has to be broken down into a number of co-dependent steps over a period of time. For instance, a user may first search on possible destinations, timeline, events, etc. After deciding when and where to go, the user may then search for the most suitable arrangements for air tickets, rental cars, lodging, meals, etc. Each step requires one or more queries, and each query results in one or more clicks on relevant pages.

One important step towards enabling services and features that can help users during their complex search quests online is the capability to identify and group related queries together. Recently, some of the major search engines have introduced a new "Search History" feature, which allows users to track their online searches by recording their queries and clicks. For example, Figure 1 illustrates a portion of a user's history as it is shown by the Bing search engine on February of 2010. This history includes a sequence of four queries displayed in reverse chronological order together with their corresponding clicks. In addition to viewing their search history, users can manipulate it by manually editing and organizing related queries and clicks into groups, or by sharing them with their friends. While these features are helpful, the manual efforts involved can be disruptive and will be untenable as the search history gets longer over time.

In fact, identifying groups of related queries has applications beyond helping the users to make sense and keep track of queries and clicks in their search history. First and foremost, query grouping allows the search engine to better understand a user's session and potentially tailor that user's search experience according to her needs. Once query groups have been identified, search engines can have a good representation of the search context behind the current query using queries and clicks in the corresponding query group. This will help to improve the quality of key components of search engines such as query suggestions, result ranking, query alterations, sessionization, and collaborative search. For example, if a search engine knows that a current query "financial statement" belongs to a {"bank of america", "financial statement"} query group, it can boost the rank of the page that provides information about how to get a Bank of America statement instead of the Wikipedia article on "financial statement", or the pages related to financial statements from other banks.

Query grouping can also assist other users by promoting task-level collaborative search. For instance, given a set of query groups created by expert users, we can select the ones that are highly relevant to the current user's query activity and recommend them to her. Explicit collaborative search can also be performed by allowing users in a trusted community to find, share and merge

- *H. Hwang is with Samsung Advanced Institute of Technology, Yongin-si, Gyeonggi-do, 446-712 South Korea; E-mail: heasooh@gmail.com.*
- *H. W. Lauw is with Institute for Infocomm Research, 1 Fusionopolis Way #21-01 Connexis (South Tower), Singapore 138632; E-mail: hwlauw@i2r.a-star.edu.sg.*
- *L. Getoor is with the Department of Computer Science, University of Maryland, AV Williams Bldg, Rm 3217 College Park, MD 20742, USA; E-mail: getoor@cs.umd.edu.*
- *A. Ntoulas is with Microsoft Research, Silicon Valley, 1065 La Avenida St, SVC-6/1040, Mountain View CA 94043, USA; E-mail: antoulas@microsoft.com.*
- *This work was done while H. Hwang, H. W. Lauw, and L. Getoor were at Microsoft Research, Silicon Valley.*


your recent history

save to ▾ | Share Windows Live / Facebook / Email

Select the results you want to save or share

Select All | None

barbados hotel x

Barbados Hotels Listings by Name x
http://barbados.org/hotname.htm add note

snorkeling x

How to Snorkel - Snorkeling Basics x
http://www.snorkeling.info/snorkeling-basics/index.html add note

How to snorkel x
http://www.sailhawaii.com/howtosnorkel.html add note

hybrid saturn vue x

saturn vue x

Saturn Vue - Wikipedia, the free encyclopedia x
http://en.wikipedia.org/wiki/Saturn_Vue add note

Saturn VUE SUV, VUE Sports Utility Vehicle, VUE warranty x
http://www.saturn.com/vue/ add note

Fig. 1. Example of search history feature in Bing.

relevant query groups to perform larger, long-term tasks on the Web.

In this paper, we study the problem of organizing a user's search history into a set of *query groups* in an automated and dynamic fashion. Each query group is a collection of queries by the same user that are relevant to each other around a common informational need. These query groups are dynamically updated as the user issues new queries, and new query groups may be created over time. To better illustrate our goal, we show in Figure 2(a) a set of queries from the activity of a real user on the Bing search engine over the period of one day, together with the corresponding query groups in Figure 2(b): the first query group contains all the queries that are related to saturn automobiles. The other groups respectively pertain to barbados vacation, sprint phone, financials, and wii game console.

Organizing the query groups within a user's history is challenging for a number of reasons. First, related queries may not appear close to one another, as a search task may span days or even weeks. This is further complicated by the interleaving of queries and clicks from different search tasks due to users' multi-tasking [3], opening multiple browser tabs, and frequently changing search topics. For instance, in Figure 2(a), the related queries "hybrid saturn vue" and "saturn dealers" are separated by many unrelated queries. This limits the effectiveness of approaches relying on time or sequence to identify related queries. Second, related queries may not be textually similar. For example, in Figure 2(b), the related queries "tripadvisor barbados" and "caribbean cruise" in Group 2 have no words in common. Therefore, relying solely on string similarity is also insufficient. Finally, as users may also manually alter their respective query groups, any automated query grouping has to respect the manual efforts or edits by the users.

To achieve more effective and robust query grouping, we do not rely solely on textual or temporal properties of queries. Instead, we leverage search behavioral data as captured within a commercial search engine's log. In particular, we develop an online query grouping method over the *query fusion* graph that combines a probabilistic *query reformulation* graph, which captures the relationship between queries frequently issued together by the users, and a *query click* graph, which captures the relationship between queries frequently leading to clicks on similar URLs. Related to our problem, are the problems of session identification [4], [5] and query clustering [6], [7] that have also used similar graphs in the past. We extend previous work in two ways. First, we use information from both the query reformulation graph and the query click graph in order to better capture various important signals of query relevance. Second, we follow an unsupervised approach where we do not require training data to bootstrap our model.

In this paper, we make the following contributions:

- We motivate and propose a method to perform query grouping in a dynamic fashion. Our goal is to ensure good performance while avoiding disruption of existing user-defined query groups.
- We investigate how signals from search logs such as query reformulations and clicks can be used together to determine the relevance among query groups. We study two potential ways of using clicks in order to enhance this process: (1) by fusing the query reformulation graph and the query click graph into a single graph that we refer to as the *query fusion graph*, and (2) by expanding the query set when computing relevance to also include other queries with similar clicked URLs.
- We show through comprehensive experimental evaluation the effectiveness and the robustness of our proposed search log-based method, especially when combined with approaches using other signals such as text similarity.

The rest of the paper is organized as follows. In Section 2, we state the goal of our paper, identifying query groups in a search history, more formally and provide an overview of our solution. In Section 3, we discuss how we can construct the query reformulation graph and the query click graph from search logs, and how to use them to determine relevance between queries or query groups within a user's history. In Section 4, we describe our algorithm to perform query grouping using the notion of relevance based on search logs. In Section 5, we present our experimental evaluation results. In Section 6, we review the related work and we conclude with a discussion on our results and future research directions in Section 7.

2 Preliminaries

Goal. Our goal is to automatically organize a user's search history into query groups, each containing one or more related queries and their corresponding clicks. Each query group corresponds to an atomic information

Time	Query
10:51:48	saturn vue
10:52:24	hybrid saturn vue
10:59:28	snorkeling
11:12:04	barbados hotel
11:17:23	sprint slider phone
11:21:02	toys r us wii
11:40:27	best buy wii console
12:32:42	financial statement
12:22:22	wii gamestop
12:59:12	saturn dealers
13:03:34	saturn hybrid review
16:34:09	bank of america
17:52:49	caribbean cruise
19:22:13	gamestop discount
19:25:49	used games wii
19:50:12	tripadvisor barbados
20:11:56	expedia
20:44:01	sprint latest model cell phones

(a) User's Search History

Group 1
saturn vue
hybrid saturn vue
saturn dealers
saturn hybrid review

Group 2
snorkeling
barbados hotel
caribbean cruise
tripadvisor barbados
expedia

Group 3
sprint slider phone
sprint latest model cell phones

Group 4
financial statement
bank of america

Group 5
toys r us wii
best buy wii console
wii gamestop
gamestop discount
used games wii

(b) Query Groups

Fig. 2. Search history of a real user over the period of one day together with the query groups

need that may require a small number of queries and clicks related to the same search goal. For example, in the case of navigational queries, a query group may involve as few as one query and one click (e.g., "cnn" and www.cnn.com). For broader informational queries, a query group may involve a few queries and clicks (e.g., Group 5 queries in Figure 2(b) are all about where to buy Wii console and games). This definition of query groups follows closely the definition of search goals given in [4].

Definition 2.1 Query Group. *A query group is an ordered list of queries, q_i, together with the corresponding set of clicked URLs, clk_i of q_i. A query group is denoted as $s = \langle\{q_1, clk_1\}, \ldots, \{q_k, clk_k\}\rangle$.*

The specific formulation of our problem is as follows:

Given: a set of existing query groups of a user, $S = \{s_1, s_2, \ldots, s_n\}$, and her current query and clicks, $\{q_c, clk_c\}$,

Find: the query group for $\{q_c, clk_c\}$, which is either one of the existing query groups in S that is most related to, or a new query group $s_c = \{q_c, clk_c\}$ if there does not exist a query group in S that is not sufficiently related to $\{q_c, clk_c\}$.

Below, we will motivate the dynamic nature of this formulation, and give an overview of the solution. The core of the solution is a measure of relevance between two queries (or query groups). We will further motivate the need to go beyond baseline relevance measures that rely on time or text, and instead propose a relevance measure based on signals from search logs.

Dynamic Query Grouping. One approach to the identification of query groups is to first treat every query in a user's history as a singleton query group, and then merge these singleton query groups in an iterative fashion (in a k-means or agglomerative way [8]). However, this is impractical in our scenario for two reasons. First, it may have the undesirable effect of changing a user's existing query groups, potentially undoing the user's own manual efforts in organizing her history. Second, it involves a high computational cost, since we would have to repeat a large number of query group similarity computations for every new query.

As in online clustering algorithms [9], we perform the grouping in a similar dynamic fashion, whereby we first place the current query and clicks into a singleton query group $s_c = \{q_c, clk_c\}$, and then compare it with each *existing* query group s_i within a user's history (i.e., $s_i \in S$). The overall process of identifying query groups is presented in Figure 3. Given s_c, we determine if there are existing query groups sufficiently relevant to s_c. If so, we merge s_c with the query group s having the highest similarity τ_{max} above or equal to the threshold τ_{sim}. Otherwise, we keep s_c as a new singleton query group and insert it into S.

Query (or Query Group) Relevance. To ensure that each query group contains closely related and relevant queries and clicks, it is important to have a suitable relevance measure sim between the current query singleton group s_c and an existing query group $s_i \in S$. There are a number of possible approaches to determine the relevance between s_c and s_i. Below, we outline a number of different relevance metrics that we will later use as baselines in experiments (see Section 5). We will also discuss the pros and cons of such metrics as well as our proposed approach of using search logs (see Section 3).

Time. One may assume that s_c and s_i are somehow relevant if the queries appear close to each other in time in the user's history. In other words, we assume that users generally issue very similar queries and clicks within a short period of time. In this case, we define the following time-based relevance metric sim_{time} that can be used in place of sim in Figure 3.

Definition 2.2 Time. $sim_{time}(s_c, s_i)$ *is defined as the inverse of the time interval (e.g. in seconds) between the times*

```
SelectBestQueryGroup
Input:
  1) the current singleton query group $s_c$ containing the
  current query $q_c$ and set of clicks $clk_c$
  2) a set of existing query groups $S = \{s_1, \ldots, s_m\}$
  3) a similarity threshold $\tau_{sim}$, $0 \leq \tau_{sim} \leq 1$
Output: The query group $s$ that best matches $s_c$, or a
new one if necessary
( 0) $s = \emptyset$
( 1) $\tau_{max} = \tau_{sim}$
( 2) for $i = 1$ to $m$
( 3)    if $sim(s_c, s_i) > \tau_{max}$
( 4)        $s = s_i$
( 5)        $\tau_{max} = sim(s_c, s_i)$
( 6)    if $s = \emptyset$
( 7)        $S = S \cup s_c$
( 8)        $s = s_c$
( 9) return s
```

Fig. 3. Algorithm for selecting the query group that is the most similar to the given query and clicked URLs.

that q_c and q_i are issued, as follows:

$$sim_{time}(s_c, s_i) = \frac{1}{|time(q_c) - time(q_i)|}$$

The queries q_c and q_i are the most recent queries in s_c and s_i respectively. Higher sim_{time} values imply that the queries are temporally closer.

Text. On a different note, we may assume that two query groups are similar if their queries are textually similar. Textual similarity between two sets of words can be measured by metrics such as the fraction of overlapping words (*Jaccard* similarity [10]) or characters (*Levenshtein* similarity [11]). We can thus define the following two text-based relevance metrics that can be used in place of sim in Figure 3.

Definition 2.3 Jaccard. *$sim_{jaccard}(s_c, s_i)$ is defined as the fraction of common words between q_c and q_i as follows:*

$$sim_{jaccard}(s_c, s_i) = \frac{|words(q_c) \cap words(q_i)|}{|words(q_c) \cup words(q_i)|}$$

Definition 2.4 Levenshtein. *$sim_{edit}(s_c, s_i)$ is defined as $1 - dist_{edit}(q_c, q_i)$. The edit distance $dist_{edit}$ is the number of character insertions, deletions, or substitutions required to transform one sequence of characters into another, normalized by the length of the longer character sequence (see [11] for more details.)*

Although the above time-based and text-based relevance metrics may work well in some cases, they cannot capture certain aspects of query similarity. For instance, sim_{time} assumes that a query is *always* followed by a related query. However, this may not be the case when the user is multi-tasking (i.e., having more than one tabs open in her browser, or digressing to an irrelevant topic and then resuming her searches). Similarly, the text-based metrics, $sim_{jaccard}$ and sim_{edit}, can capture the relevance between query groups around textually similar queries such as "ipod" and "apple ipod", but will fail to identify relevant query groups around queries such as "ipod" and "apple store", since they are not textually similar. Additionally, the text-based metrics may mistakenly identify query groups around, say, "jaguar car manufacturer" and "jaguar animal reserve" as relevant, since they share some common text.

Therefore, we need a relevance measure that is robust enough to identify similar query groups beyond the approaches that simply rely on the textual content of queries or time interval between them. Our approach makes use of search logs in order to determine the relevance between query groups more effectively. In fact, the search history of a large number of users contains signals regarding query relevance, such as which queries tend to be issued closely together (query reformulations), and which queries tend to lead to clicks on similar URLs (query clicks). Such signals are user-generated and are likely to be more robust, especially when considered at scale. We suggest measuring the relevance between query groups by exploiting the query logs and the click logs simultaneously. We will discuss our proposed relevance measure in greater detail in Sections 3 and 4.

In fact, the idea of making use of signals in query logs to measure similarity between queries has been explored in previous work, although not to the same extent as our proposed approach. Here, we outline two such methods, *CoR* and *ATSP*, which will also be compared against in our experimental section (see Section 5).

CoR. Co-Retrieval (or CoR) is based on the principle that a pair of queries are similar if they tend to retrieve similar pages on a search engine. This approach is similar to the ones discussed in[12], [13].

Definition 2.5 CoR. *$sim_{cor}(s_c, s_i)$ is the Jaccard coefficient of q_c's set of retrieved pages $retrieved(q_c)$ and q_i's set of retrieved pages $retrieved(q_i)$ and is defined as:*

$$sim_{cor}(s_c, s_i) = \frac{|retrieved(q_c) \cap retrieved(q_i)|}{|retrieved(q_c) \cup retrieved(q_i)|}$$

Unlike [12] which relies on textual comparison, we compare two queries based on the overlap in pages retrieved. We consider a page to be retrieved by a search engine if it has not only been shown to some users, but has also been clicked at least once in the past one year. Notice that this is a stronger definition that favors CoR as a baseline because of the relevance signals in the form of clicks. Differently from our approach, CoR makes use of neither reformulation signals (whether one query frequently follows another) nor click signals (whether queries frequently lead to clicks on similar pages).

ATSP. This technique is based on the principle that two queries issued in succession in the search logs are closely related. In [5], the authors present a solution that first reorders a sequence of user queries to group similar queries together by solving an instance of the Asymmetric Traveler Salesman Problem (ATSP). Once the queries are reordered, query groups are generated by determining "cut points" in the chain of queries, i.e. two successive queries whose similarity is less than a threshold η. Note that ATSP needs to operate on the whole set of queries that we are interested in grouping as it involves an initial reordering step.

Definition 2.6 ATSP *$sim_{ATSP}(s_c, s_i)$ is defined as the number of times two queries, q_c and q_i, appear in succession in the search logs over the number of times q_c appears. More formally:*

$$sim_{ATSP}(s_c, s_i) = \frac{freq(q_c, q_i)}{freq(q_c)}$$

In our work we consider both query pairs having common clicked URLs and the query reformulations through a combined query fusion graph.

3 Query Relevance Using Search Logs

We now develop the machinery to define the *query relevance* based on Web search logs. Our measure of relevance is aimed at capturing two important properties of relevant queries, namely: (1) queries that frequently appear together as reformulations and (2) queries that have induced the users to click on similar sets of pages. We start our discussion by introducing three search behavior graphs that capture the aforementioned properties. Following that, we show how we can use these graphs to compute query relevance and how we can incorporate the clicks following a user's query in order to enhance our relevance metric.

3.1 Search Behavior Graphs

We derive three types of graphs from the search logs of a commercial search engine. The *query reformulation graph*, $\mathcal{QRG}$, represents the relationship between a pair of queries that are likely reformulations of each other. The *query click graph*, $\mathcal{QCG}$, represents the relationship between two queries that frequently lead to clicks on similar URLs. The *query fusion graph*, $\mathcal{QFG}$, merges the information in the previous two graphs. All three graphs are defined over the same set of vertices $\mathcal{V_Q}$, consisting of queries which appear in at least one of the graphs, but their edges are defined differently.

3.1.1 Query Reformulation Graph

One way to identify relevant queries is to consider *query reformulations* that are typically found within the query logs of a search engine. If two queries that are issued consecutively by many users occur frequently enough, they are likely to be reformulations of each other. To measure the relevance between two queries issued by a user, the time-based metric, sim_{time}, makes use of the interval between the timestamps of the queries within the user's search history. In contrast, our approach is defined by the statistical frequency with which two queries appear next to each other in the entire query log, over all of the users of the system.

To this end, based on the query logs, we construct the *query reformulation graph*, $\mathcal{QRG} = (\mathcal{V_Q}, \mathcal{E_{QR}})$, whose set of edges, $\mathcal{E_{QR}}$, are constructed as follows: for each query pair (q_i, q_j), where q_i is issued before q_j within a user's day of activity, we count the number of such occurrences across all users' daily activities in the query logs, denoted $count_r(q_i, q_j)$. Assuming infrequent query pairs are not good reformulations of each other, we filter out infrequent pairs and include only the query pairs whose counts exceed a threshold value, τ_r. For each (q_i, q_j) with $count_r(q_i, q_j) \geq \tau_r$, we add a directed edge from q_i to q_j to $\mathcal{E_{QR}}$. The edge weight, $w_r(q_i, q_j)$, is defined as the normalized count of the query transitions:

$$w_r(q_i, q_j) := \frac{count_r(q_i, q_j)}{\sum_{(q_i, q_k) \in \mathcal{E_{QR}}} count_r(q_i, q_k)}.$$

3.1.2 Query Click Graph

A different way to capture relevant queries from the search logs is to consider queries that are likely to induce users to click frequently on the same set of URLs. For example, although the queries "ipod" and "apple store" do not share any text or appear temporally close in a user's search history, they are relevant because they are likely to have resulted in clicks about the ipod product. In order to capture such property of relevant queries, we construct a graph called the query click graph, $\mathcal{QCG}$.

We first start by considering a bipartite *click-through graph*, $\mathcal{CG} = (\mathcal{V_Q} \cup \mathcal{V_U}, \mathcal{E_C})$, used by Fuxman et al. [14]. $\mathcal{CG}$ has two distinct sets of nodes corresponding to queries, $\mathcal{V_Q}$, and URLs, $\mathcal{V_U}$, extracted from the click logs. There is an edge $(q_i, u_k) \in \mathcal{E_C}$, if query q_i was issued and URL u_k was clicked by some users. We weight each edge (q_i, u_k) by the number of times q_i was issued and u_k was clicked, $count_c(q_i, u_k)$. As before, we filter out infrequent pairs using a threshold τ_c. In this way, using the $\mathcal{CG}$, we identify pairs of queries that frequently lead to clicks on similar URLs.

Next, from $\mathcal{CG}$, we derive our *query click graph*, $\mathcal{QCG} = (\mathcal{V_Q}, \mathcal{E_{QC}})$, where the vertices are the queries, and a directed edge from q_i to q_j exists if there exists at least one URL, u_k, that both q_i and q_j link to in $\mathcal{CG}$. The weight of edge (q_i, q_j) in $\mathcal{QCG}$, $w_c(q_i, q_j)$, is defined as the weighted asymmetric Jaccard similarity [10] as follows:

$$w_c(q_i, q_j) = \frac{\sum_{u_k} min(count_c(q_i, u_k), count_c(q_j, u_k))}{\sum_{u_k} count_c(q_i, u_k)}$$

This captures the intuition that q_j is more related to q_i if more of q_i's clicks fall on the URLs that are also clicked for q_j.

3.1.3 Query Fusion Graph

The query reformulation graph, $\mathcal{QRG}$, and the query click graph, $\mathcal{QCG}$, capture two important properties of relevant queries respectively. In order to make more effective use of both properties, we combine the query reformulation information within $\mathcal{QRG}$ and the query-click information within $\mathcal{QCG}$ into a single graph, $\mathcal{QFG} = (\mathcal{V_Q}, \mathcal{E_{QF}})$, that we refer to as the *query fusion graph*. At a high level, $\mathcal{E_{QF}}$ contains the set of edges that exist in either $\mathcal{E_{QR}}$ or $\mathcal{E_{QC}}$. The weight of edge (q_i, q_j) in $\mathcal{QFG}$, $w_f(q_i, q_j)$, is taken to be a linear sum of the edge's weights, $w_r(q_i, q_j)$ in $\mathcal{E}_{\mathcal{QR}}$ and $w_c(q_i, q_j)$ in $\mathcal{E}_{\mathcal{QC}}$, as follows:

$$w_f(q_i, q_j) = \alpha \times w_r(q_i, q_j) + (1 - \alpha) \times w_c(q_i, q_j)$$

The relative contribution of the two weights is controlled by α, and we denote a query fusion graph constructed with a particular value of α as $\mathcal{QFG}(\alpha)$. The effects of varying α is explored further in Section 5.

3.2 Computing Query Relevance

Having introduced the search behavior graphs in the previous section, we now compute the relevance between two queries. More specifically, for a given user query q, we compute a relevance vector using $\mathcal{QFG}$, where each entry corresponds to the relevance value of each query $q_j \in \mathcal{V}_{\mathcal{Q}}$ to q.

The edges in $\mathcal{QFG}$ correspond to pairs of relevant queries extracted from the query logs and the click logs. However, it is not sufficiently effective to use the pairwise relevance values directly expressed in $\mathcal{QFG}$ as our query relevance scores. Let us consider a vector $\mathbf{r_q}$, where each entry, $\mathbf{r_q}(q_j)$, is $w_f(q, q_j)$ if there exists an edge from q to q_j in $\mathcal{QFG}$, and 0 otherwise. One straightforward approach for computing the relevance of q_j to q is to use this $\mathbf{r_q}(q_j)$ value. However, although this may work well in some cases, it will fail to capture relevant queries that are not directly connected in $\mathcal{QFG}$ (and thus $\mathbf{r_q}(q_j) = 0$).

Therefore, for a given query q, we suggest a more generic approach of obtaining query relevance by defining a Markov chain for q, $\mathcal{MC}_q$, over the given graph, $\mathcal{QFG}$, and computing the stationary distribution of the chain. We refer to this stationary distribution as the *fusion relevance vector* of q, $\mathbf{rel_q^F}$, and use it as a measure of query relevance throughout this paper.

In a typical scenario, the stationary probability distribution of $\mathcal{MC}_q$ can be estimated using the matrix multiplication method, where the matrix corresponding to $\mathcal{MC}_q$ is multiplied by itself iteratively until the resulting matrix reaches a fixpoint. However, given our setting of having thousands of users issuing queries and clicks in real-time and the huge size of $\mathcal{QFG}$, it is infeasible to perform the expensive matrix multiplication to compute the stationary distribution whenever a new query comes in. Instead, we pick the most efficient Monte Carlo random walk simulation method among the ones presented in [15], and use it on $\mathcal{QFG}$ to approximate the stationary distribution for q. Figure 4 outlines our algorithm.

The algorithm in Figure 4 computes the fusion relevance vector of a given query q, $\mathbf{rel_q^F}$. It requires the following inputs in addition to $\mathcal{QFG}$. First, we introduce a jump vector of q, $\mathbf{g_q}$, that specifies the probability that a query is selected as a starting point of a random walk. Since we set $\mathbf{g_q}(q')$ to 1 if $q' = q$, and 0 otherwise, q will always be selected; in the next section we will generalize $\mathbf{g_q}$ to have multiple starting points by considering both q and the clicks for q. A *damping factor*, $d \in [0, 1]$ (similar to the original PageRank algorithm [16]), determines the probability of random walk re-start at each node.

```
Relevance(q)
Input:
  1) the query fusion graph, $\mathcal{QFG}$
  2) the jump vector, $g$
  3) the damping factor, $d$
  4) the total number of random walks, $numRWs$
  5) the size of neighborhood, $maxHops$
  6) the given query, $q$
Output: the fusion relevance vector for $q$, $\mathbf{rel_q^F}$
( 0) Initialize $\mathbf{rel_q^F} = \mathbf{0}$
( 1) $numWalks = 0; numVisits = 0$
( 2) while $numWalks < numRWs$
( 3)   $numHops = 0; v = q$
( 4)   while $v \neq NULL \wedge numHops < maxHops$
( 5)     $numHops{+}{+}$
( 6)     $\mathbf{rel_q^F}(v){+}{+}; numVisits{+}{+}$
( 7)     $v = SelectNextNodeToVisit(v)$
( 8)   $numWalks{+}{+}$
( 9) For each $v$, normalize $\mathbf{rel_q^F}(v) = \mathbf{rel_q^F}(v)/numVisits$
```

Fig. 4. Algorithm for calculating the query relevance by simulating random walks over the query fusion graph.

Two additional inputs control the accuracy and the time budget of the random walk simulation: the total number of random walks, $numRWs$, and the size of neighborhood explored, $maxHops$. As $numRWs$ increases, the approximation accuracy of the fusion relevance vector improves by the law of large numbers. We limit the length of each random walk to $maxHops$, assuming that a transition from q to q' is very unlikely if no user in the search logs followed q by q' in less than $maxHops$ number of intermediate queries. In practice, we typically use $numRWs = 1,000,000$ and $maxHops = 5$, but we can reduce the number of random walk samples or the lengths of random walks by decreasing both parameters for a faster computation of $\mathbf{rel_q^F}$.

The random walk simulation then proceeds as follows: we use the jump vector $\mathbf{g_q}$ to pick the random walk starting point. At each node v, for a given damping factor d, the random walk either continues by following one of the outgoing edges of v with a probability of d, or stops and re-starts at one of the starting points in $\mathbf{g_q}$ with a probability of $(1-d)$. Then, each outgoing edge, (v, q_i), is selected with probability $w_f(v, q_i)$, and the random walk always re-starts if v has no outgoing edge. The selection of the next node to visit based on the outgoing edges of the current node v in $\mathcal{QFG}$ and the damping factor d is performed by the $SelectNextNodeToVisit$ process in Step (7) of the algorithm, which is illustrated in Figure 5. Notice that each random walk simulation is independent of another, so can be parallelized.

After simulating $numRWs$ random walks on the $\mathcal{QFG}$ starting from the node corresponding to the given query q, we normalize the number of visits of each node by the number of all the visits, finally obtaining $\mathbf{rel_q^F}$, the fusion relevance vector of q. Each entry of the vector, $\mathbf{rel_q^F}(q')$, corresponds to the fusion relevance score of a query $q' \in \mathcal{V}_{\mathcal{Q}}$ to the given query q. It is the probability that q' node is visited along a random walk originated from q node over the $\mathcal{QFG}$.

Lastly, we show that there exists a unique fusion rel-

```
SelectNextNodeToVisit(v)
Input:
  1) the query fusion graph, $\mathcal{QFG}$
  2) the jump vector, $g$
  3) the damping factor, $d$
  4) the current node, $v$
Output: the next node to visit, $q_i$
( 0) if $random() < d$
( 1)   $V = \{q_i \mid (v, q_i) \in \mathcal{E}_{\mathcal{QF}}\}$
( 2)   pick a node $q_i \in V$ with probability $w_f(v, q_i)$
( 3) else
( 4)   $V = \{q_i \mid g(q_i) > 0\}$
( 5)   pick a node $q_i \in V$ with probability $g(q_i)$
( 6) return $q_i$
```

Fig. 5. Algorithm for selecting the next node to visit.

evance vector of a given query q, $\mathbf{rel_q^F}$. It is well-known that for a finite ergodic Markov chain, there exists a unique stationary distribution. In fact, the random walk simulation algorithm described in Figure 4 approximates $\mathbf{rel_q^F}$ that corresponds to the stationary distribution of the Markov chain for q, $\mathcal{MC}_q$. To prove the uniqueness of $\mathbf{rel_q^F}$, it is sufficient to show that $\mathcal{MC}_q$ is ergodic.

Given a query q and a damping factor d, the Markov chain for q, $\mathcal{MC}_q$, is defined as follows: first, the finite state space of $\mathcal{MC}_q$, denoted Ω_q, contains all the queries reachable from the given query q in $\mathcal{QFG}$ ($\Omega_q \subset \mathcal{V_Q}$). Then, we define the transition matrix of $\mathcal{MC}_q$. For each state q_i and q_j in Ω_q, the transition probability from state q_i to state q_j, $\mathcal{MC}_q(q_i, q_j)$, is defined as

$$\mathcal{MC}_q(q_i, q_j) = \begin{cases} d * w_f(q_i, q_j) & \text{if } q_j \neq q, \\ d * w_f(q_i, q_j) + (1-d) & \text{if } q_j = q. \end{cases}$$

If q_i has no outgoing edge in $\mathcal{QFG}$, we set $\mathcal{MC}_q(q_i, q_j)$ to 1 for the next state $q_j = q$ and 0 otherwise. Also note that if q_i and q_j are not directly connected in $\mathcal{QFG}$, $w_f(q_i, q_j) = 0$. As in Boldi et al. [17], we assume that the transition matrix of $\mathcal{MC}_q$ is aperiodic. Also, each state in Ω_q has a positive transition probability to state q (actually, $\mathcal{MC}_q(q_i, q) \geq 1 - d \ \forall q_i \in \Omega_q$), so any state in $\mathcal{MC}_q$ can reach any other state in $\mathcal{MC}_q$ through state q. Thus, $\mathcal{MC}_q$ is ergodic, which guarantees the existence of unique stationary distribution of $\mathcal{MC}_q$. However, we want to mention that $\mathcal{MC}_q$ is a conceptual model, and we do not materialize $\mathcal{MC}_q$ for each query q in $\mathcal{QFG}$ to calculate $\mathbf{rel_q^F}$ in practice. Instead, for a given query q, we simply adjust edge weights in $\mathcal{QFG}$ accordingly, and set state q as the start state of every random walk to ensure that only states of $\mathcal{MC}_q$ among nodes in $\mathcal{QFG}$ are visited.

3.3 Incorporating Current Clicks

In addition to query reformulations, user activities also include clicks on the URLs following each query submission. The clicks of a user may further help us infer her search interests behind a query q and thus identify queries and query groups relevant to q more effectively. In this section, we explain how we can use the click information of the current user to expand the random walk process to improve our query relevance estimates. Note that the approach we introduce in this section is independent of modeling the query click information as $\mathcal{QCG}$ in Section 3.1.2 to build $\mathcal{QFG}$. Here, we use clicks of the current user to better understand her search intent behind the currently issued query, while clicks of massive users in the click logs are aggregated into $\mathcal{QCG}$ to capture the degree of relevance of query pairs through commonly clicked URLs.

We give a motivating example that illustrates why it may be helpful to take into account clicked URLs of q to compute the query relevance. Let us consider that a user submitted a query "jaguar". If we compute the relevance scores of each query in $\mathcal{V_Q}$ with respect to the given query only, both the queries related to the car "jaguar" and those related to the animal "jaguar" get high fusion relevance scores. This happens because we do not know the actual search interest of the current user when she issues the query "jaguar". However, if we know the URLs clicked by the current user following the query "jaguar" (e.g. the Wikipedia article on animal "jaguar"), we can infer the search interest behind the current query and assign query relevance scores to queries in $\mathcal{V_Q}$ accordingly. In this way, by making use of the clicks, we can give much higher query relevance scores to queries related to "animal jaguar" than those related to "car jaguar". This idea of biasing the random walks towards a certain subset of the graph nodes is similar in spirit to topic-sensitive PageRank [18].

We now describe how we use the clicked URLs by the current user together with the given query q to better capture her search intent. First, we identify the set of URLs, *clk*, that were clicked by the current user after issuing q. Then, we use *clk* and the click-through graph $\mathcal{CG}$ to expand the space of queries considered when we compute the fusion relevance vector of q. Unlike the jump vector $\mathbf{g_q}$ in Section 3.2 that reflects the given query q only, we now consider both q and *clk* together when we set a new jump vector.

Given q and *clk*, we employ a *click jump vector*, $\mathbf{g_{clk}}$, that represents the queries in $\mathcal{CG}$ that have also induced clicks to the URLs within *clk*. Each entry in $\mathbf{g_{clk}}$, $\mathbf{g_{clk}}(q_i)$, corresponds to the relevance of query q_i to the URLs in *clk*. Using $\mathcal{CG}$, we define $\mathbf{g_{clk}}$ as the proportion of the number of clicks to *clk* induced by q_i ($q_i \in \mathcal{V_Q} \backslash \{q\}$) to the total number of clicks to *clk* induced by all the queries in $\mathcal{V_Q} \backslash \{q\}$.

$$\mathbf{g_{clk}}(q_i) := \frac{\sum_{u_k \in clk} count_c(q_i, u_k)}{\sum_{q_j \in \mathcal{V_Q}, q_j \neq q} \sum_{u_k \in clk} count_c(q_j, u_k)}$$

Since the given query q is already captured in $\mathbf{g_q}$, we set the entry in $\mathbf{g_{clk}}$ corresponding to q to 0 ($\mathbf{g_{clk}}(q) = 0$).

Now, we introduce a new jump vector $\mathbf{g_{(q,clk)}}$ that considers both q and *clk* by incorporating $\mathbf{g_{clk}}$ that biases the random jump probabilities towards queries related to the clicks, *clk*. In particular, we combine $\mathbf{g_q}$ and $\mathbf{g_{clk}}$ by defining $\mathbf{g_{(q,clk)}}$ as the weighted sum of $\mathbf{g_q}$ in Section 3.2 and the click jump vector $\mathbf{g_{clk}}$. We control the importance of query and click by using w_{query} and

w_{click} ($w_{query} + w_{click} = 1$), thus $\mathbf{g}_{(\mathbf{q},\mathbf{clk})}(q) = w_{query}$ and $\mathbf{g}_{(\mathbf{q},\mathbf{clk})}(q') = w_{click} * \mathbf{g}_{\mathbf{clk}}(q')$ for every query $q' \in \mathcal{V}_{\mathcal{Q}} \backslash \{q\}$. Once $\mathbf{g}_{(\mathbf{q},\mathbf{clk})}$ is set, we simulate random walks and estimate the fusion relevance vector in a similar way as before, with one difference. Notice that in Section 3.2, when calculating $\mathbf{rel}^{\mathbf{F}}_{\mathbf{q}}$, all the random walks start from the node corresponding to q, because $\mathbf{g}_{\mathbf{q}}(q)$ is the only nonzero entry in the jump vector $\mathbf{g}_{\mathbf{q}}$ ($\mathbf{g}_{\mathbf{q}}(q) = 1$). Now, however, the random walk simulation can start from any query node q' for which $\mathbf{g}_{(\mathbf{q},\mathbf{clk})}(q') > 0$, with a probability of $\mathbf{g}_{(\mathbf{q},\mathbf{clk})}(q')$. We denote this alternate query fusion vector obtained from $\mathbf{g}_{(\mathbf{q},\mathbf{clk})}$ as $\mathbf{rel}^{\mathbf{F}}_{(\mathbf{q},\mathbf{clk})}$.

In the following sections, fusion relevance vectors, $\mathbf{rel}^{\mathbf{F}}_{\mathbf{q}}$ and $\mathbf{rel}^{\mathbf{F}}_{(\mathbf{q},\mathbf{clk})}$, are referred to as $\mathbf{rel}_{\mathbf{q}}$ and $\mathbf{rel}_{(\mathbf{q},\mathbf{clk})}$ respectively, assuming that we, by default, use the query fusion graph $\mathcal{QFG}$, not $\mathcal{QRG}$ or $\mathcal{QCG}$, to compute relevance vectors.

4 Query Grouping using the $\mathcal{QFG}$

In this section, we outline our proposed similarity function sim_{rel} to be used in the online query grouping process outlined in Section 2. For each query, we maintain a *query image*, which represents the relevance of other queries to this query. For each query group, we maintain a *context vector*, which aggregates the images of its member queries to form an overall representation. We then propose a similarity function sim_{rel} for two query groups based on these concepts of context vectors and query images. Note that our proposed definitions of query reformulation graph, query images, and context vectors are crucial ingredients, which lend significant novelty to the Markov chain process for determining relevance between queries and query groups.

Context Vector. For each query group, we maintain a *context vector* which is used to compute the similarity between the query group and the user's latest singleton query group. The *context vector* for a query group s, denoted $\mathbf{cxt_s}$, contains the relevance scores of each query in $\mathcal{V}_{\mathcal{Q}}$ to the query group s, and is obtained by aggregating the fusion relevance vectors of the queries and clicks in s. If s is a singleton query group containing only $\{q_{s_1}, clk_{s_1}\}$, it is defined as the fusion relevance vector $\mathbf{rel}_{(\mathbf{q_{s_1}},\mathbf{clk_{s_1}})}$. For a query group $s = \langle\{q_{s_1}, clk_{s_1}\}, \ldots, \{q_{s_k}, clk_{s_k}\}\rangle$ with $k > 1$, there are a number of different ways to define $\mathbf{cxt_s}$. For instance, we can define it as the fusion relevance vector of the most recently added query and clicks, $\mathbf{rel}_{(\mathbf{q_{s_k}},\mathbf{clk_{s_k}})}$. Other possibilities include the average or the weighted sum of all the fusion relevance vectors of the queries and clicks in the query group. In our experiments, we calculate the context vector of a query group s by weighting the queries and the clicks in s by recency, as follows:

$$\mathbf{cxt_s} = w_{recency} \sum_{j=1}^{k} (1 - w_{recency})^{k-j} \mathbf{rel}_{(\mathbf{q_{s_j}},\mathbf{clk_{s_j}})}$$

Note that if $\{q_{s_k}, clk_{s_k}\}$ are the most recent query and clicks added to the query group, this can be rewritten:

$$\mathbf{cxt_s} = w_{recency} * \mathbf{rel}_{(\mathbf{q_{s_k}},\mathbf{clk_{s_k}})} + (1 - w_{recency})\mathbf{cxt_{s'}}$$

where $s' = \langle\{q_{s_1}, clk_{s_1}\}, \ldots, \{q_{s_{k-1}}, clk_{s_{k-1}}\}\rangle$. In our implementation we used $w_{recency} = 0.3$.

Query Image. The fusion relevance vector of a given query q, $\mathbf{rel_q}$, captures the degree of relevance of each query $q' \in \mathcal{V}_{\mathcal{Q}}$ to q. However, we observed that it is not effective or robust to use $\mathbf{rel_q}$ itself as a relevance measure for our online query grouping. For instance, let us consider two relevant queries, "financial statement" ("fs") and "bank of america" ("boa"), in Figure 2(b). We may use the relevance value in the fusion relevance vectors, $\mathbf{rel}_{\text{“}\mathbf{fs}\text{''}}(\text{“}boa\text{''})$ or $\mathbf{rel}_{\text{“}\mathbf{boa}\text{''}}(\text{“}fs\text{''})$. Usually, however, it is a very tiny number that does not comprehensively express the relevance of the search tasks of the queries, thus is not an adequate relevance measure for an effective and robust online query grouping. Instead, we want to capture the fact that both queries highly pertain to financials.

To this end, we introduce a new concept, the *image* of q, denoted $\mathbf{I}(q)$, that expresses q as the set of queries in $\mathcal{V}_{\mathcal{Q}}$ that are considered highly relevant to q. We generate $\mathbf{I}(q)$ by including every query q' whose relevance value to q, $\mathbf{rel_q}(q')$, is within top-X percentage. To do this, we sort the queries by relevance, and find the cutoff such that the sum of the relevance values of the most relevant queries accounts for $X\%$ of the total probability mass. We break ties randomly. In our experiments, $X = 99\%$. We found that even with this high percentage, the size of the image of the query is typically very small compared to the total number of possible queries in $\mathcal{QFG}$. The image of a query group s, $\mathbf{I}(s)$, is defined in the same way as $\mathbf{I}(q)$ except that the context vector of s, $\mathbf{cxt_s}$, is used in the place of $\mathbf{rel}_{(\mathbf{q},\mathbf{clk})}$.

Now, we define the relevance metric for query groups, sim_{rel} ($\in [0,1]$), based on $\mathcal{QFG}$. Two query groups are similar if their common image occupies high probability mass in both of the context vectors of the query groups. We use the above definitions of context vector and query image to capture this intuition.

Definition 4.1 *$sim_{rel}(s_1, s_2)$, the relevance between two query groups s_1 and s_2, is defined as follows:*

$$sim_{rel}(s_1, s_2) = \sum_{q \in \mathbf{I}(s_1) \cap \mathbf{I}(s_2)} \mathbf{cxt_{s_1}}(q) * \sum_{q \in \mathbf{I}(s_1) \cap \mathbf{I}(s_2)} \mathbf{cxt_{s_2}}(q)$$

Then, the relevance between the user's latest singleton query group $s_c = \{q_c, clk_c\}$ and an existing query group $s_i \in S$ will be

$$sim_{rel}(s_c, s_i) = \sum_{q \in \mathbf{I}(s_c) \cap \mathbf{I}(s_i)} \mathbf{rel}_{(\mathbf{q_c},\mathbf{clk_c})}(q) * \sum_{q \in \mathbf{I}(s_c) \cap \mathbf{I}(s_i)} \mathbf{cxt_{s_i}}(q).$$

The relevance metric sim_{rel} is used in the Step (5) of the algorithm in Figure 3 in place of sim. In this way, the latest singleton query group s_c will be attached to the query group s that has the highest similarity sim_{rel}.

Online Query Grouping. The similarity metric that we described in Definition 4.1 operates on the images of a query and a query group. Some applications such as query suggestion may be facilitated by fast on-the-fly grouping of user queries. For such applications, we can avoid performing the random walk computation of fusion relevance vector for every new query in real-time, and instead pre-compute and cache these vectors for some queries in our graph. This works especially well for the popular queries. In this case, we are essentially trading off disk storage for run-time performance. We estimate that to cache the fusion relevance vectors of 100 million queries, we would require disk storage space in the hundreds of gigabytes. This additional storage space is insignificant relative to the overall storage requirement of a search engine. Meanwhile, retrieval of fusion relevance vectors from the cache can be done in milliseconds. Hence, for the remainder of this paper, we will focus on evaluating the effectiveness of the proposed algorithms in capturing query relevance.

5 Experiments

5.1 Experimental Setup

In this section, we study the behavior and performance of our algorithms on partitioning a user's query history into one or more groups of related queries. For example, for the sequence of queries "caribbean cruise"; "bank of america"; "expedia"; "financial statement", we would expect two output partitions: first, {"caribbean cruise", "expedia"} pertaining to travel-related queries, and, second, {"bank of america", "financial statement"} pertaining to money-related queries.

Data. To this end, we obtained the query reformulation and query click graphs by merging a number of monthly search logs from a commercial search engine. Each monthly snapshot of the query log adds approximately 24% new nodes and edges in the graph compared to the exactly preceding monthly snapshot, while approximately 92% of the mass of the graph is obtained by merging 9 monthly snapshots. To reduce the effect of noise and outliers, we pruned the query reformulation graph by keeping only query pairs that appeared at least two times ($\tau_q = 2$), and the query click graph by keeping only query-click edges that had at least ten clicks ($\tau_c = 10$). This produced query and click graphs that were 14% and 16% smaller compared to their original respective graphs. Based on these two graphs, we constructed the query fusion graph as described in Section 3 for various parameter settings of α.

In order to create test cases for our algorithms, we used the search activity (comprising at least two queries) of a set of 200 users (henceforth called the *Rand200* dataset) from our search log. To generate this set, users were picked randomly from our logs, and two human labelers examined their queries and assigned them to either an existing group or a new group if the labelers deemed that no related group was present. A user's queries were included in the *Rand200* dataset if both labelers were in agreement in order to reduce bias and subjectivity while grouping. The labelers were allowed access to the Web in order to determine if two seemingly distant queries were actually related (e.g. "alexander the great" and "gordian knot"). The average number of groups in the dataset was 3.84 with 30% of the users having queries grouped in more than 3 groups.

Performance Metric. To measure the quality of the output groupings, for each user, we start by computing query pairs in the labeled and output groupings. Two queries form a pair if they belong to the same group, with lone queries pairing with a special "null" query.

To evaluate the performance of our algorithms against the groupings produced by the labelers, we will use the Rand Index [19] metric, which is a commonly employed measure of similarity between two partitions. The Rand Index similarity between two partitions X,Y of n elements each is defined as $RandIndex(X,Y) = (a+b)/\begin{pmatrix} n \\ 2 \end{pmatrix}$, where a is the number of pairs that are in the same set in X and the same set in Y, and b is the number of pairs that are in different sets in x and in different sets in Y. Higher $RandIndex$ values indicate better ability of grouping related queries together for a given algorithm.

Default values. In the following, we will study different aspects of our proposed algorithms. Unless we explicitly specify differently, we use the following default parameters: damping factor $d = 0.6$, *top-X* $= 99\%$, $\alpha = 0.7$, click importance $w_{click} = 0.2$, recency $w_{recency} = 0.3$, and similarity threshold $\tau_{sim} = 0.9$. We have picked these values by repeating a set of experiments with varying values for these parameters and selecting the ones that would allow our algorithm to achieve the best performance on *Rand200* based on the $RandIndex$ metric. We followed the same approach for the baselines that we implemented as well. We will also evaluate the approaches on additional test sets (*Lo100*, *Me100*, *Hi100*), which will be described later. Since our method involves a random walk, we also tested for statistical significance of each configuration across runs. The results that we present in the remainder of the section are statistically significant at the 0.001 level according to the t-test statistical significance test [20] across runs.

5.2 Using Search Logs

As discussed in Section 3, our query grouping algorithm relies heavily on the use of search logs in two ways: first, to construct the query fusion graph used in computing query relevance, and, second, to expand the set of queries considered when computing query relevance. We start our experimental evaluation, by investigating how we can make the most out of the search logs.

In our first experiment, we study *how we should combine* the query graphs coming from the query reformulations and the clicks within our query log. Since combining the two graphs is captured by the α parameter as we discussed in Section 3, we evaluated our algorithm over the graphs that we constructed for increasing values

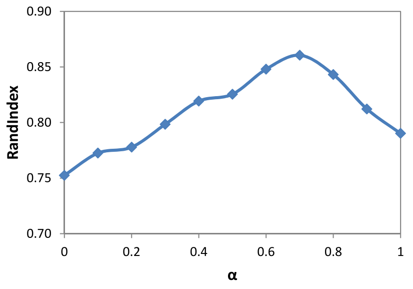


Fig. 6. Varying mix of query and click graphs.

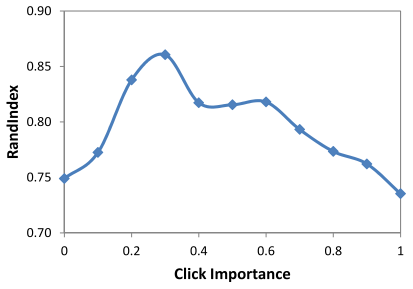


Fig. 7. Varying the click importance w_{click}.

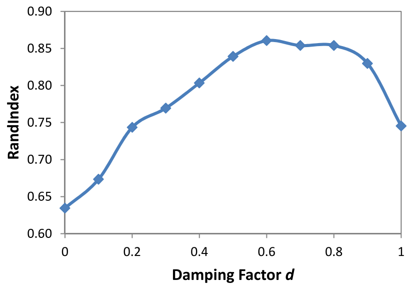


Fig. 8. Varying the damping factor d.

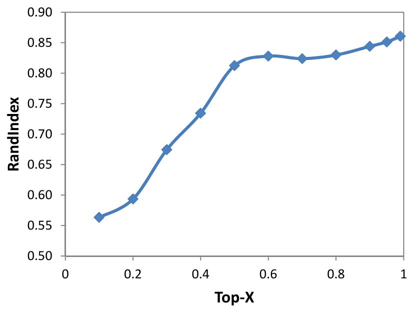


Fig. 9. Varying the fraction of related queries in Top-X.

of α. The result is shown in Figure 6; the horizontal axis represents α (i.e., how much weight we give to the query edges coming from the query reformulation graph), while the vertical axis shows the performance of our algorithm in terms of the $RandIndex$ metric. As we can see from the graph, our algorithm performs best ($RandIndex = 0.86$) when α is around 0.7, with the two extremes (only edges from clicks, i.e., $\alpha = 0.0$, or only edges from reformulations, i.e., $\alpha = 1.0$) performing lower. It is interesting to note that, based on the shape of the graph, edges coming from query reformulations are deemed to be slightly more helpful compared to edges from clicks. This is because there are 17% fewer click-based edges than reformulation-based edges, which means that random walks performed on the query reformulation graph can identify richer query images as there are more available paths to follow in the graph.

We now turn to study the *effect of expanding the query set* based on the user clicks when computing query relevance. To this end, we evaluated the performance of our algorithm for increasing values of click importance w_s and we show the result in Figure 7. Based on this figure, we observe that, in general, taking user clicks into account to expand the considered query set helps to improve performance. Performance rises up to a point ($w_{click} = 0.3$), after which it starts degrading. At the two extremes (when only queries from user clicks are used to seed the random walks, i.e., $w_s = 1$, or when only the current query is used, i.e., $w_{click} = 0$, performance is generally lower.

5.3 Varying the Parameters

Given the previous results on how to utilize the information from search logs, we now turn to studying the remaining parameters of our algorithms.

Damping Factor. The damping factor d is the probability of continuing a random walk, instead of starting over from one of the query nodes in the jump vector. As shown in Figure 8, $RandIndex$ is lower for very low damping factor, increases together with the damping factor, and maxes out damping factors between 0.6 and 0.8. This confirms our intuition that related queries are close to the current query in our query fusion graph and that they can be captured with short random walks (small d) from the current query. At the extreme where damping factor is 0, we observe a lower performance as the query image is essentially computed on a random sample from the jump vector without exploiting the link information of the query fusion graph.

Top-X. Top-X is the fraction of the sum of relevance scores of related queries that are included in the *image* of a query. As Figure 9 shows, we get better performance for very high X, such as 0.99. We pick a high X, in order to keep most of the related queries that can be potential useful for capturing query similarities. Usually, even though we use a very high X value such as 0.99, the number of related queries in a query image is still much smaller than $|\mathcal{V}_Q|$ as related queries obtain much higher relevance scores than those of irrelevant ones.

Similarity Threshold. The similarity threshold τ_{sim} helps us determine whether we should start a new group for the current query or attach to an existing one. We show how performance varies based on increasing similarity thresholds in Figure 10. In general, as the similarity threshold increases, the $RandIndex$ value becomes higher. This is expected as the higher the similarity is, the more likely that a session would include query groups containing highly related queries. A high threshold is also useful for avoiding the effect of having unrelated

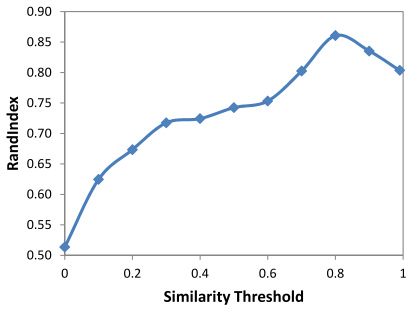


Fig. 10. Varying the similarity threshold τ_{sim}.

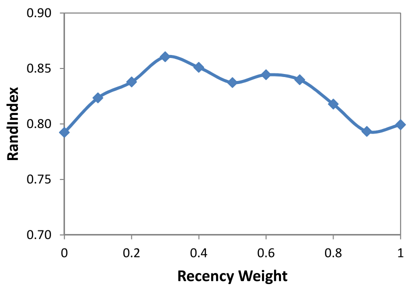


Fig. 11. Varying the recency weight $w_{recency}$.

but very popular queries (e.g., "ebay", "yahoo") that may appear frequently as reformulations of each other. As τ_{sim} increases from 0.8 to 1, the $RandIndex$ drops since such τ_{sim} is too strict to group related queries together, resulting in many small groups.

Recency Weight. We finally study the recency weight $w_{recency}$ that affects how much weight we are giving to the fusion relevance vectors within an existing query group. Larger values of $w_{recency}$ mean that we are favoring more the latest query that was assigned to a given query group. We show how performance varies based on increasing $w_{recency}$ values in Figure 11. Overall, we observe that we get the best performance for $w_{recency}$ values between 0.3 and 0.6.

5.4 Performance Comparison

We now compare the performance of our proposed methods against five different baselines. For these baselines, we use the same *SelectBestQueryGroup* as in Figure 3 with varying relevance metrics.

As the first baseline, we use a time-based method (henceforth referred to as *Time*) that groups queries

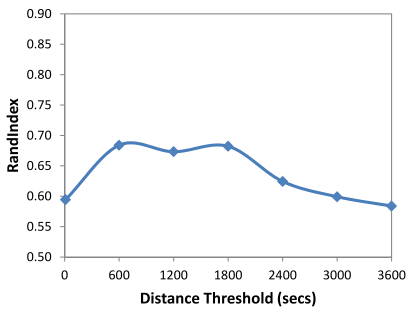


Fig. 12. Varying the time threshold.

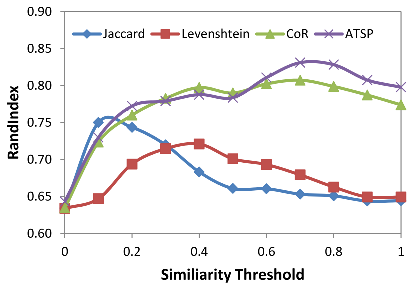


Fig. 13. Varying the similarity threshold.

based on whether the time difference between a query and the most recent previous query is above a threshold. It is essentially the same as the *Time* metric introduced in Section 2, except that instead of measuring similarity as the inverse of the time interval, we measure the distance in terms of the time interval (in seconds). Figure 12 shows the performance of this method for varying time thresholds (measured in seconds). We will use 600 secs (highest $RandIndex$ value in Figure 12) as the default threshold for this method.

The next two baselines are based on text similarity. *Jaccard* similarity uses the fraction of overlapping keywords between two queries, while *Levenshtein* similarity calculates the edit distance, normalized by the maximum length of the two queries being compared. It may capture misspellings and typographical errors that may elude the word-based Jaccard. Figure 13 shows their performance as we vary the similarity threshold. As with *Time*, the optimal performance is reached at an intermediate threshold, 0.1 (default) in the case of *Jaccard*, and 0.4 (default) for *Levenshtein*.

Our last two baselines exploit click and query graphs. More specifically, we have implemented the co-retrieval baseline (henceforth referred to as *CoR*) to assign a query to the group with the highest overlap in the retrieved results, as described in Section 2. We have also implemented the method based on the Asymmetric Traveler Salesman Problem (henceforth referred to as *ATSP*) as described in [5]. Since both of these baselines are threshold-based, we study their performance for increasing threshold values in Figure 13, and then set the similarity threshold for CoR to 0.7 (default) and for ATSP to 0.7(default).

We compare the baseline methods with our method that uses the query fusion graph. For our method (denoted as $\mathcal{QFG}$), we use the default parameters that we specified in Section 5.1. We report the results on the *Rand200* dataset in the first row of Table 1, where we use bold-face to denote the best performance for a dataset (we will discuss the remaining rows in the next section). Overall, *Time* and *Levenshtein* perform worse than the rest of the algorithms. This is an indication that the queries issued by the users are interleaved in terms of their topics (hence *Time* performs badly) and also that the edit distance between queries is not able to capture related queries too well. *Jaccard* is performing slightly better than these two but it also cannot capture the groupings very well, with the *CoR* method coming next.

	Time	*Leven-shtein*	*Jaccard*	*CoR*	*ATSP*	$\mathcal{QFG}$
Rand200	0.683	0.721	0.750	0.807	0.831	**0.860**
Lo100	0.620	0.732	0.762	0.794	**0.832**	0.821
Me100	0.632	0.712	0.748	0.802	0.857	**0.868**
Hi100	0.654	0.729	0.742	0.809	0.871	**0.882**

TABLE 1
Comparative performance (RandIndex) of our methods. Best performance in each dataset is shown in bold.

Finally, our $\mathcal{QFG}$ method and the *ATSP* method perform the best with $\mathcal{QFG}$ performing slightly better than *ATSP*.

The techniques that we have studied so far fall into different categories and attempt to capture different aspects of query similarities; *Time* simply looks at the time intervals, *Jaccard* and *Levenshtein* exploit textual similarities of queries, while *CoR*, *ATSP* and $\mathcal{QFG}$ use the search logs. Therefore, given the different natures of these algorithms it is reasonable to hypothesize that they do well for different kinds of queries. In particular, since our $\mathcal{QFG}$ method relies on the accurate estimation of a query image within the query fusion graph, it is expected to perform better when the estimation was based on more information and is therefore more accurate. On the other hand, if there are queries that are rare in the search logs or do not have many outgoing edges in our graph to facilitate the random walk, the graph-based techniques may perform worse due to the lack of edges. We study how the structure of the graph affects the performance of the algorithms as follows.

5.5 Varying Graph Connectivity

In order to better estimate the query transition probabilities in our query fusion graph, it is helpful to have as much usage information encoded in the graph as possible. More specifically, if the queries within a user's session are issued more frequently, they are also more likely to have more outgoing edges in the graph and thus facilitate the random walks going out of these queries. At the same time, more popular queries will have more accurate counts in the graph and this may lead to higher confidence when we compute the query images.

To gain a measure of usage information for a given user, we look at the average outdegree of the user's queries (average outdegree), as well as the average counts among the outgoing links (average weight) in the query reformulation graph. In order to study the effects of usage information on the performance of our algorithms, we created three additional test sets of 100 users each. The sets were also manually labeled as we described in Section 5.1. The first set, *Lo100* contains the search activity of 100 users, with average outdegree < 5 and average weight < 5. Similarly, *Me100* contains user activity for users having $5 \leq$ average outdegree < 10 and $5 \leq$ average weight < 10, while *Hi100* contains user activity with average outdegree ≥ 10 and average weight ≥ 10.

Based on these datasets, we evaluate again the performance of our algorithms and we show the results in the bottom three lines of Table 1. As we can see from the table, for $\mathcal{QFG}$, subsets with higher usage information also tend to have higher $RandIndex$ values. *Hi100* ($RandIndex$ = 0.88) performs better than *Me100* ($RandIndex$ = 0.868), which in turn outperforms *Lo100* ($RandIndex$ = 0.821). *ATSP* shows a similar trend (higher usage shows better performance) and it outperforms $\mathcal{QFG}$ at the *Lo100* dataset. *CoR*'s performance is more or less similar for the different datasets which is expected as it does not use the graphs directly. For *Jaccard*, it is most efficient when the connectivity around the queries within a user's session is relatively low. We do not observe any significant difference in the performance of the other baselines (*Time* and *Levenshtein*) in these new datasets.

Overall, we observe that different techniques might be more appropriate for different degrees of usage information (and hence connectivity) of the graph. Higher connectivity implies that the queries are well known and may be well-connected in the graph, while lower connectivity might imply that the query is new or not very popular. Since our goal is to have a good performance accross the board for all queries we study the combination of these methods next.

5.6 Combining the Methods

The results of the previous experiment point out the contrast between the performance of the different methods. This suggests that a combination of two methods may yield better performance than either method individually. We explore combining two methods by merging the output query groups as follows: given the output groups of any two methods, query pairs that belong to a group within one *or* within the other, will belong to the same group in the combined output.

Table 2 shows the performance gained by combining $\mathcal{QFG}$ with each baseline. For $\mathcal{QFG}$+*Jaccard* and $\mathcal{QFG}$+*Levenshtein*, the combination performs better than the individual methods. $\mathcal{QFG}$+*Time* performs better than *Time* but worse than $\mathcal{QFG}$.

Interestingly, for $\mathcal{QFG}$+*Jaccard*, we now get a more consistent performance across the three test sets (*Lo100*, *Me100*, *Hi100*) at around 0.89. The biggest boost to $\mathcal{QFG}$'s performance is obtained for *Lo100*; it is more than for *Me100* or *Hi100*. This result is noteworthy as it implies that the combination method performs gracefully across queries for which we may have different usage information in the graph. Combining $\mathcal{QFG}$ with *CoR* and *ATSP* improves slightly their performance but not as much as the combination of $\mathcal{QFG}$ and *Jaccard*. This is mostly due to the fact that *CoR* captures similar information as the click portion of the $\mathcal{QFG}$, while *ATSP* captures similar information to the query reformulation portion of $\mathcal{QFG}$.

In summary, from the experimental results, we observe that using the click graph in addition to query reformulation graph in a unified query fusion graph helps improve performance. Additionally, the query fusion graph performs better for queries with higher usage

	$\mathcal{QFG}$+ Time	$\mathcal{QFG}$+ Levenshtein	$\mathcal{QFG}$+ Jaccard	$\mathcal{QFG}$+ CoR	$\mathcal{QFG}$+ ATSP
Rand200	0.722	0.794	**0.894**	0.832	0.864
Lo100	0.692	0.812	**0.899**	0.839	0.864
Me100	0.699	0.800	**0.909**	0.846	0.872
Hi100	0.732	0.821	**0.914**	0.867	0.888

TABLE 2
Performance (RandIndex) of combined methods. Best performance in each dataset is shown in bold.

information and handily beats time-based and keyword similarity-based baselines for such queries. Finally, keyword similarity-based methods help complement our method well providing for a high and stable performance regardless of the usage information.

6 Related Work

While we are not aware of any previous work that has the same objective of organizing user history into query groups, there has been prior work in determining whether two queries belong to the same search task. In recent work, Jones and Klinkner [4] and Boldi et al. [5] investigate the search-task identification problem. More specifically, Jones and Klinkner [4] considered a search session to consist of a number of tasks (missions), and each task further consists of a number of sub-tasks (goals). They trained a binary classifier with features based on time, text, and query logs to determine whether two queries belong to the same task. Boldi et al. [5] employed similar features to construct a query flow graph, where two queries linked by an edge were likely to be part of the same search mission.

Our work differs from these prior works in the following aspects. First, the query-log based features in [4], [5] are extracted from co-occurrence statistics of query pairs. In our work, we additionally consider query pairs having common clicked URLs and we exploit both co-occurrence and click information through a combined query fusion graph. [4] will not be able to break ties when an incoming query is considered relevant to two existing query groups. Additionally, our approach does not involve learning and thus does not require manual labeling and re-training as more search data come in; our Markov random walk approach essentially requires maintaining an updated query fusion graph. Finally, our goal is to provide users with useful query groups on-the-fly while respecting existing query groups. On the other hand, search task identification is mostly done at server side with goals such as personalization, query suggestions [5] etc.

Some prior work also looked at the problem of how to segment a user's query streams into "sessions". In most cases, this segmentation was based on a "timeout threshold" [21], [22], [23], [24], [25], [26], [27]. Some of them, such as [23], [26], looked at the segmentation of a user's browsing activity, and not search activity. Silverstein et al. [27] proposed a timeout threshold value of five minutes, while others [21], [22], [24], [25] used various threshold values. As shown in Section 5, time is not a good basis for identifying query groups, as users may be multitasking when searching online [3], thus resulting in interleaved query groups.

The notion of using text similarity to identify related queries has been proposed in prior work. He et al. [24] and Ozmutlu and Çavdur [28] used the overlap of terms of two queries to detect changes in the topics of the searches. Lau and Horvitz [29] studied the different refinement classes based on the keywords in queries, and attempted to predict these classes using a Bayesian classifier. Radlinski and Joachims [30] identified query sequences (called chains) by employing a classifier that combines a timeout threshold with textual similarity features of the queries, as well as the results returned by those queries. While text similarity may work in some cases, it may fail to capture cases where there is "semantic" similarity between queries (e.g. "ipod" and "apple store") but no textual similarity. In Section 5, we investigate how we can use textual similarity to complement approaches based on search logs to obtain better performance.

The problem of online query grouping is also related to query clustering [13], [31], [6], [7], [32]. The authors in [13] found query clusters to be used as possible questions for a FAQ feature in an Encarta reference Web site by relying on both text and click features. In Beeferman and Berger [6] and Baeza-Yates and Tiberi [7], commonly clicked URLs on query-click bipartite graph are used to cluster queries. The authors in [31] defined clusters as bicliques in the click graph. Unlike online query grouping, the queries to be clustered are provided in advance, and might come from many different users. The query clustering process is also a batch process that can be accomplished offline. While these prior work make use of click graphs, our approach is much richer in that we use the click graph in combination with the reformulation graph, and we also consider indirect relationships between queries connected beyond one hop in the click graph. This problem is also related to document clustering [33], [34], with the major difference being the focus on clustering queries (only a few words) as compared to clustering documents for which term distributions can be estimated well.

Graphs based on query and click logs [35] have also been used in previous work for different applications such as query suggestions [5], query expansion [36], ranking [37] and keyword generation [14]. In several cases, variations of random walks have been applied on the graph in order to identify the most important nodes. In Craswell and Szummer [37], a Markov random walk was applied on the click graph to improve ranking. In Fuxman et al. [14], a random walk was applied on the click-through graph to determine useful keywords; while in Collins-Thomson and Callan [36], a random walk was applied for query suggestion/expansion with the node having the highest stationary probability being the best candidate for suggestion. As we discussed in Section 3, we take advantage of the stationary probabilities computed from the graph as a descriptive vector (image) for each query in order to determine similarity among query groups.

7 Conclusion

The query reformulation and click graphs contain useful information on user behavior when searching online. In this paper, we show how such information can be used effectively for the task of organizing user search histories into query groups. More specifically, we propose combining the two graphs into a query fusion graph. We further show that our approach that is based on probabilistic random walks over the query fusion graph outperforms time-based and keyword similarity-based approaches. We also find value in combining our method with keyword similarity-based methods, especially when there is insufficient usage information about the queries. As future work, we intend to investigate the usefulness of the knowledge gained from these query groups in various applications such as providing query suggestions and biasing the ranking of search results.

References

[1] J. Teevan, E. Adar, R. Jones, and M. A. S. Potts, "Information re-retrieval: repeat queries in yahoo's logs," in *SIGIR*. New York, NY, USA: ACM, 2007, pp. 151–158.
[2] A. Broder, "A taxonomy of web search," *SIGIR Forum*, vol. 36, no. 2, pp. 3–10, 2002.
[3] A. Spink, M. Park, B. J. Jansen, and J. Pedersen, "Multitasking during Web search sessions," *Information Processing and Management*, vol. 42, no. 1, pp. 264–275, 2006.
[4] R. Jones and K. L. Klinkner, "Beyond the session timeout: Automatic hierarchical segmentation of search topics in query logs," in *CIKM*, 2008.
[5] P. Boldi, F. Bonchi, C. Castillo, D. Donato, A. Gionis, and S. Vigna, "The query-flow graph: Model and applications," in *CIKM*, 2008.
[6] D. Beeferman and A. Berger, "Agglomerative clustering of a search engine query log," in *KDD*, 2000.
[7] R. Baeza-Yates and A. Tiberi, "Extracting semantic relations from query logs," in *KDD*, 2007.
[8] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.
[9] W. Barbakh and C. Fyfe, "Online clustering algorithms," *International Journal of Neural Systems*, vol. 18, no. 3, pp. 185–194, 2008.
[10] M. Berry and M. Browne, Eds., *Lecture Notes in Data Mining*. World Scientific Publishing Company, 2006.
[11] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," *Soviet Physics Doklady*, vol. 10, p. 707, 1966.
[12] M. Sahami and T. D. Heilman, "A web-based kernel function for measuring the similarity of short text snippets," in *WWW '06: Proceedings of the 15th international conference on World Wide Web*. New York, NY, USA: ACM, 2006, pp. 377–386.
[13] J.-R. Wen, J.-Y. Nie, and H.-J. Zhang, "Query clustering using user logs," *ACM Transactions in Information Systems*, vol. 20, no. 1, pp. 59–81, 2002.
[14] A. Fuxman, P. Tsaparas, K. Achan, and R. Agrawal, "Using the wisdom of the crowds for keyword generation," in *WWW*, 2008.
[15] K. Avrachenkov, N. Litvak, D. Nemirovsky, and N. Osipova, "Monte carlo methods in PageRank computation: When one iteration is sufficient," *SIAM Journal on Numerical Analysis*, vol. 45, no. 2, pp. 890–904, 2007.
[16] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: Bringing order to the web," in *Technical report, Stanford University*, 1998.
[17] P. Boldi, M. Santini, and S. Vigna, "Pagerank as a function of the damping factor," in *WWW*, 2005.
[18] T. H. Haveliwala, "Topic-sensitive PageRank," in *WWW*, 2002.
[19] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical Association*, vol. 66, no. 336, pp. 846–850, 1971.
[20] D. D. Wackerly, W. M. III, and R. L. Scheaffer, *Mathematical Statistics with Applications*, sixth edition ed. Duxbury Advanced Series, 2002.
[21] P. Anick, "Using terminological feedback for web search refinement: A log-based study," in *SIGIR*, 2003.
[22] B. J. Jansen, A. Spink, C. Blakely, and S. Koshman, "Defining a session on Web search engines: Research articles," *Journal of the American Society for Information Science and Technology*, vol. 58, no. 6, pp. 862–871, 2007.
[23] L. D. Catledge and J. E. Pitkow, "Characterizing browsing strategies in the World-Wide Web," *Computer Networks and ISDN Systems*, vol. 27, no. 6, pp. 1065–1073, 1995.
[24] D. He, A. Goker, and D. J. Harper, "Combining evidence for automatic Web session identification," *Information Processing and Management*, vol. 38, no. 5, pp. 727–742, 2002.
[25] R. Jones and F. Diaz, "Temporal profiles of queries," *ACM Transactions on Information Systems*, vol. 25, no. 3, p. 14, 2007.
[26] A. L. Montgomery and C. Faloutsos, "Identifying Web browsing trends and patterns," *Computer*, vol. 34, no. 7, pp. 94–95, 2001.
[27] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz, "Analysis of a very large Web search engine query log," *SIGIR Forum*, vol. 33, no. 1, pp. 6–12, 1999.
[28] H. C. Ozmutlu and F. Çavdur, "Application of automatic topic identification on Excite Web search engine data logs," *Information Processing and Management*, vol. 41, no. 5, pp. 1243–1262, 2005.
[29] T. Lau and E. Horvitz, "Patterns of search: Analyzing and modeling Web query refinement," in *UM*, 1999.
[30] F. Radlinski and T. Joachims, "Query chains: Learning to rank from implicit feedback," in *KDD*, 2005.
[31] J. Yi and F. Maghoul, "Query clustering using click-through graph," in *WWW*, 2009.
[32] E. Sadikov, J. Madhavan, L. Wang, and A. Halevy, "Clustering query refinements by user intent," in *WWW*, 2010.
[33] T. Radecki, "Output ranking methodology for document-clustering-based boolean retrieval systems," in *SIGIR*. New York, NY, USA: ACM, 1985, pp. 70–76.
[34] V. R. Lesser, "A modified two-level search algorithm using request clustering," in *Report No. ISR-11 to the National Science Foundation, Section VII, Department of Computer Science, Cornell University*, 1966.
[35] R. Baeza-Yates, "Graphs from search engine queries," *Theory and Practice of Computer Science (SOFSEM)*, vol. 4362, pp. 1–8, 2007.
[36] K. Collins-Thompson and J. Callan, "Query expansion using random walk models," in *CIKM*, 2005.
[37] N. Craswell and M. Szummer, "Random walks on the click graph," in *SIGIR*, 2007.

Heasoo Hwang received her PhD degree from the University of California at San Diego. Her main research interests include effective and efficient search over large-scale graph-structured data. She is a research staff member at Samsung Advanced Institute of Technology.

Hady W. Lauw is a researcher at the Institute for Infocomm Research in Singapore. Previously, he was a postdoctoral researcher at Microsoft Research Silicon Valley. He earned a doctorate degree in computer science at Nanyang Technological University in 2008 on an A*STAR graduate fellowship.

Lise Getoor is an associate professor at the University of Maryland, College Park. Her research interests include machine learning and reasoning under uncertainty, with applications to information integration, database management, and social media. She has a PhD in Computer Science from Stanford University.

Alexandros Ntoulas is a researcher at Microsoft Research, Silicon Valley. His research interests include systems and algorithms that facilitate the monitoring, collection, management, mining, and searching of information on the Web. He has a PhD in Computer Science from the University of California, Los Angeles.