The query-flow graph: model and applications

Paolo Boldi[1*] boldi@dsi.unimi.it
Francesco Bonchi[2] bonchi@yahoo-inc.com
Carlos Castillo[2] chato@yahoo-inc.com
Debora Donato[2] debora@yahoo-inc.com
Aristides Gionis[2] gionis@yahoo-inc.com
Sebastiano Vigna[1*] vigna@dsi.unimi.it

[1]DSI, Università degli Studi di Milano, Italy
[2]Yahoo! Research Labs Barcelona, Spain

ABSTRACT

Query logs record the queries and the actions of the users of search engines, and as such they contain valuable information about the interests, the preferences, and the behavior of the users, as well as their implicit feedback to search-engine results. Mining the wealth of information available in the query logs has many important applications including query-log analysis, user profiling and personalization, advertising, query recommendation, and more.

In this paper we introduce the *query-flow graph*, a graph representation of the interesting knowledge about latent querying behavior. Intuitively, in the query-flow graph a directed edge from query q_i to query q_j means that the two queries are likely to be part of the same "search mission". Any path over the query-flow graph may be seen as a searching behavior, whose likelihood is given by the strength of the edges along the path.

The query-flow graph is an outcome of query-log mining and, at the same time, a useful tool for it. We propose a methodology that builds such a graph by mining time and textual information as well as aggregating queries from different users. Using this approach we build a real-world query-flow graph from a large-scale query log, and we demonstrate its utility in concrete applications, namely, *finding logical sessions*, and *query recommendation*. We believe, however, that the usefulness of the query-flow graph goes beyond these two applications.

1. INTRODUCTION

The huge volume of information recorded daily in query logs contains a wealth of valuable knowledge about how web users interact with search engines as well as information about the interests and the preferences of those users. Extracting behavioral patterns from this wealth of information is a key step towards improving the service provided by search engines and towards developing innovative web-search paradigms. Unfortunately, mining query logs poses many technical challenges that arise due to the very large volume of data, the high level of noise, poorly formulated queries, ambiguity, and sparsity, among others.

In this paper we introduce the concept of the *query-flow graph*, which is a graph modeling user behavioral patterns and query dependencies. The query-flow graph is an actionable, aggregated representation of the interesting information contained in a large query-log. In particular, the phenomenon of interest is the *sequentiality of similar queries*: the fundamental two dimensions that drive the construction of the query-flow graph are the temporal order of queries and their similarity.

Given a query log, the nodes of the query-flow graph are all the queries contained in the log, and a directed edge between two queries q_i, q_j has a weight $w(q_i, q_j)$ representing the probability that the two queries are part of the same search mission, and that they appear in the given order. Thus when $w(q_i, q_j)$ is high, we may think of q_j as a typical reformulation of q_i, thus a step ahead towards the successful completion of a possible search mission.

The main contribution of this paper is introducing the query-flow graph and providing a methodology for constructing such a graph based on mining query logs. Besides this, we demonstrate the usefulness of the query-flow graph in two applications: finding logical sessions and query recommendation.

With respect to finding logical sessions, we allow them to be intertwined, thus modeling the behavior of users who have a number of interests/goals and submit queries related to the information needs of those interests/goals but in an interleaved fashion. We also address this problem starting from the entire query history of users and not from timeout-driven sessions. To our knowledge, this is the first time that the modeling of the problem of finding query chains allows such complexity. We formulate the problem of finding intertwined query chains as an asymmetric traveling salesman problem (ATSP), which we approximate with a greedy heuristic.

For the problem of query recommendation we propose an algorithm that builds on the concept of query-flow graph and allows leveraging not only similarity between queries but the overall complex structure in a neighborhood of the graph. Our recommendation algorithm is based on performing a random walk with restart to the original query of the user or to a small set of queries representing the recent querying history.

*Part of this work was done while the authors were visiting Yahoo! Research Labs, Barcelona

Submitted for confidential review.
Last updated: June 4, 2008.

This paper is summarized as follows. Section 2 is an overview of the related work. In Section 3 we define our notation and concepts and in Section 4 we discuss our algorithm for constructing the query-flow graph. Then we describe two applications: finding query chains in 5, and query recommendations in Section 6. Finally, Section 7 includes a few concluding remarks.

2. RELATED WORK

Query logs are widely considered as a very rich source of knowledge on user behavior. The main challenge in analyzing query logs lies in extracting interesting relations from the raw lists of user actions. Many different approaches have been proposed in order to discover essential features or hidden relations in query logs.

Query graphs. One main research line attempts to infer the hidden semantics of user interactions with search engines by projecting the data over different types of graphs. Baeza-Yates [1], identifies five different types of graphs. In all cases, the nodes are queries; a link is introduced between two nodes respectively if: (*i*) the queries contain the same word(s) (*word graph*), (*ii*) the queries belong to the same session (*session graph*), (*iii*) users clicked on the same urls in the list of their results (*url cover graph*), (*iv*) there is a link between the two clicked urls (*url link graph*) (*v*) there are l common terms in the content of the two urls (*link graph*). In [1], it is suggested that one application of these graphs is session segmentation which is one of the applications we study in this paper.

Baeza-Yates and Tiberi [2], study a weighted version of the *cover graph*. Their analysis provides information not only about how people query but also about how they behave after a query and the content distribution of what they look at. Moreover the authors study several characteristics of *click graphs*, i.e., bipartite graphs of queries and urls, where a query and a url are connected if a user clicked on a url that was an answer for a query. This framework is used to infer semantic relations among queries and to detect *multitopical urls*, i.e., urls that cover either several topics or a single very general topic.

Query recommendation. Query recommendation is a core task for large industrial search engines. Most of the work on query recommendation is focused on measures of query similarity [20, 10] that can be used for query expansion [3] or query clustering [3, 19]. A first attempt to model the users' sequential search behavior is presented by Zhang and Nasraoui [20]: the arcs between consecutive queries in the same session are weighted by a dumping factor d, meanwhile the similarity values for non consecutive queries are calculated by multiplying the values of arcs that join them. Instead, Fonseca et al. [10] discover related queries with a method based on association rules. Each transaction in the query log is seen as a session in which a single user submits a sequence of related queries in a time interval. Their notion of session is similar to the one we use in this paper.

Reference [3] studies the problem of suggesting related queries issued by other users and query expansion methods to construct artificial queries. Their method is used to recommend queries that are related to the input query but may search for different issues. The clustering is based on a term-weight vector representation of queries, obtained from the aggregation of the term-weight vectors of the urls clicked after the query. Wen et al. [19] also present a clustering method for query recommendation that is centered around four notions of query distance: the first notion is based on keywords or phrases of the query; the second on string matching of keywords; the third on common clicked urls; and the fourth on the distance of the clicked documents in some pre-defined hierarchy.

Query Segmentation. Segmenting the query stream into sets of related information-seeking queries, i.e., *logical sessions*, has many applications: apart for query recommendation, since logical session can help in understanding the relationship between queries given the user intent, they are valuable for user profiling and personalization. He and Göker [11] studied different timeouts to segment user sessions, and later extended their work [12] to consider other features such as the overlap between terms in two consecutive queries. Radlinski and Joachims [16] observe that users often perform a sequence, or chain, of queries with a similar information need; they refer to this sequence of reformulated queries as *query chains*. Their paper presents a simple method for automatically detecting query chains in query and clickthrough logs and show how to learn better retrieval functions using evidence of query chains. Recently the problem of query session detection was also considered by Jones and Klinkner [14] where a method for automated segmentation is proposed and evaluated.

Temporal classification. Considering time features might have other applications beyond segmenting query stream. Jones and Diaz [13] introduce a model to measure the distribution of documents retrieved in response to a query, over the time domain, in order to create a temporal profile for a query. They show that such a temporal profile can provide valuable information about the likely quality of query results.

Random walk models. Craswell and Szummer [8] describe a Markov random walk model for ranking documents. A backward random walk is performed over the click graph, leading to a method for retrieving relevant documents that have not yet been clicked for a predefined query and rank those effectively. The random walk we introduce is performed over a completely different graph and with the objective of ranking queries instead of documents. Collins-Thompson and Callan [7] use a Markov random model for query expansion. Their setting is also different from ours: the stationary distribution of the model is used to obtain probability estimates that a potential expansion term reflects aspects of the original query.

3. BASIC CONCEPTS

In this section we provide the basic idea behind the query-flow graph. In summary the query-flow graph is an usage-oriented, actionable, compact representation of the information contained in a query log, and it is aimed at facilitating the analysis of user behavior.

Query log. A query log records information about the *search actions* of the users of a search engine. Such information includes the queries submitted by the users, documents viewed as a result to each query, and documents clicked by the users. A typical query log $\mathscr{L}$ is a set of records $\langle q_i, u_i, t_i, V_i, C_i \rangle$, where: q_i is the submitted query, u_i is an anonymized identifier for the user who submitted the query, t_i is a timestamp, V_i is the set of documents returned as

results to the query, and C_i is the set of documents clicked by the user.

In the above representation, we assume that if $\mathscr{U}$ is the set of users to the search engine and $\mathscr{D}$ is the set of documents indexed by the search engine, then $u_i \in \mathscr{U}$ and $C_i \subseteq V_i \subseteq \mathscr{D}$. For the purposes of this paper, we do not use any information from the results of the queries (C_i and V_i)—we are only mentioning them above for completeness. Thus, subsequently we denote query logs by $\mathscr{L} = \{ \langle q_i, u_i, t_i \rangle \}$.

Sessions. A user query session, or session, is defined as the sequence of queries of one particular user within a specific time limit. More formally, if t_θ is a timeout threshold, a user query session S is a *maximal* ordered sequence

$$S = \big\langle \langle q_{i_1}, u_{i_1}, t_{i_1} \rangle, \ldots, \langle q_{i_k}, u_{i_k}, t_{i_k} \rangle \big\rangle,$$

where $u_{i_1} = \cdots = u_{i_k} = u \in \mathscr{U}$, $t_{i_1} \leq \cdots \leq t_{i_k}$, and $t_{i_{j+1}} - t_{i_j} \leq t_\theta$, for all $j = 1, 2, \ldots, k-1$.

Given a query log $\mathscr{L}$, the corresponding set of sessions can be constructed by sorting all records of the query log first by userid u_i, and then by timestamp t_i, and by performing one additional pass to split sessions of the same user whenever the time difference of two queries exceeds the timeout threshold. Whenever we used a timeout threshold for splitting sessions, we set $t_\theta = 30$ minutes, as this is the typical timeout that is often used in web log analysis [6, 18, 15].

Supersessions. The sequence of all the queries of a user in the querylog, ordered by timestamp, is called a *supersession*. Thus, a supersession is a sequence of sessions in which consecutive sessions have time difference larger than t_θ.

Chains. A chain is a topically coherent sequence of queries of one user. Radlinski and Joachims [16] defined a chain as *"a sequence of queries with a similar information need"*. For instance, a query chain may contain the following sequence of queries [14]: "`brake pads`"; "`auto repair`"; "`auto body shop`"; "`batteries`"; "`car batteries`"; "`buy car battery online`". The concept of chain is also referred to in the literature with the terms *mission* [14] and *logical session* [1]. Unlike the concept of session, chains involve relating queries based on the user information need, which is an extremely hard problem, so we do not try to formally define chains here.

We note that for chains we do not impose any timeout constraint. Therefore, as an example, all the queries of a user who is interested in planning a trip to a far-away destination and searches for tickets, hotels, and other tourist information over a period of several weeks should be grouped in the same chain. Additionally, for the queries composing a chain we do not require them to be consecutive. Following the previous example, the user who is planning the far-away trip may search for tickets in one day, then make some other queries related to a newly released movie, and then return to trip planning the next day by searching for a hotel. Thus, a session may contain queries from many chains, and inversely, a chain may contain queries from many sessions.

The query-flow graph. The final concept we define is the query-flow graph, which is a central contribution in our paper. The query-flow graph G_{qf} is a directed graph $G_{\text{qf}} = (V, E, w)$ where:

- the set of nodes is $V = Q \cup \{s, t\}$, i.e., the distinct set of queries Q submitted to the search engine and two special nodes s and t, representing a *starting state* and a *terminal state* which can be seen as the begin and the end of a chain;
- $E \subseteq V \times V$ is the set of *directed* edges;
- $w : E \to (0\,..\,1]$ is a weighting function that assigns to every pair of queries $(q, q') \in E$ a weight $w(q, q')$ representing the probability that q and q' are part of the same chain.

We will mainly focus only on how to compute the weighting function w. We will see how different applications may lead to different weighting schemes.

In our setting, even if a query has been submitted multiple times to the search engine, possibly by many different users, it is anyway represented by a single node in the query-flow graph. The two special nodes s and t are used to capture the begin and the end of query chains. In other words, the existence of an edge (s, q_i) represents that q_i may be pontentially a starting query in a chain, and $w(s, q_i)$ quantifies the probability of this event happening. Similarly an edge (q_i, t) models the probability of q_i being a terminal query in a chain.

The edge weights in the query-flow graph are obtained from the query log by a machine-learning algorithm as described in the following section.

4. BUILDING THE QUERY-FLOW GRAPH

In this section we describe our approach for building the query-flow graph $G_{\text{qf}} = (V, E, w)$. Our algorithm takes as input a set of sessions $\mathscr{S}(\mathscr{L}) = \{S_1, \ldots, S_m\}$, which in our case are extracted from a query log $\mathscr{L}$ from the Yahoo! UK search engine in early 2008. As we already mentioned, the set of sessions can be easily constructed by sorting the queries by userid and by timestamp, and splitting them using the timeout threshold.

As stated in the previous section, the set of nodes V in the query-flow graph is the distinct set of queries Q in $\mathscr{L}$ plus the two special nodes s and t. The key aspect of the construction of the query-flow graph is to define the weighting function $w : E \to (0\,..\,1]$.

For the moment we leave apart the two special nodes s and t: we will discuss later about how to connect them with the other nodes of the graph. Given two queries $q, q' \in Q$ we *tentatively* connect them with an edge if there is at least one session in $\mathscr{S}(\mathscr{L})$ in which q and q' are consecutive. In other words, we form the set of tentative edges T as:

$$T = \{(q, q') \mid \exists S_j \in \mathscr{S}(\mathscr{L}) \text{ s.t. } q = q_i \in S_j \wedge q' = q_{i+1} \in S_j\}.$$

Next, we want to associate a probability $w(q, q')$ to each edge $(q, q') \in T$. We do this by building a machine learning model. The first step is to build for each edge $(q, q') \in T$ a set of features associated with the edge. Those features are computed over all sessions in $\mathscr{S}(\mathscr{L})$ that contain the queries q and q' appearing in this order and consecutively. The features we use aggregate, among other, information about the time difference in which the queries are submitted [11], textual similarity of the queries [12, 14], and the number of sessions in which they appear. We shortly describe the features in more detail.

For learning the weighting function from these features, we use training data. This training data is created by picking at random a set of edges (q, q') (excluding the edges where $q = s$ or $q' = t$) and manually assigning them a label

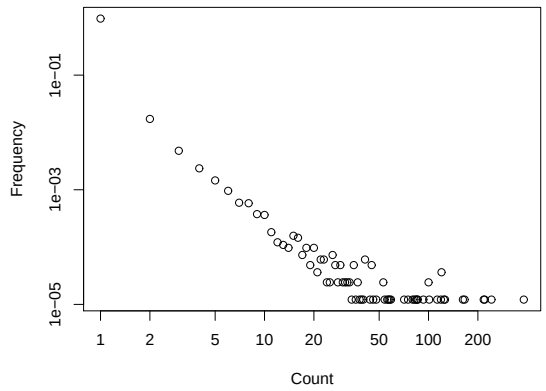

Figure 1: The distribution of counts (number of times a given pair of query appears consecutively in that order in $\mathscr{S}(\mathscr{L})$); it is a power law with a spike at 1 (most pairs being hapax).

same_chain. This label, or target variable, is assigned by human editors and is 0 if q and q' are not part of the same chain, and it is 1 if they are part of the same chain. The probability of having an edge included in the training set is proportional to the number of times the queries forming that edge occur in that order and consecutively in the query log.

We then use this training data to learn the function $w(-,-)$, given the set of features and the label for each edge in T.

The features. We built a set of 18 features to be used to compute the function $w(-,-)$ for each edge in T. Several of these features were shown to be effective for query segmentation [11, 12, 14] and can be summarized as follows:

- **Textual features.** We compute the textual similarity of queries q and q' using various similarity measures, including cosine similarity, Jaccard coefficient, and size of intersection. Those measures are computed on sets of stemmed words and on character-level 3-grams.
- **Session features.** We compute the number of sessions in which the pair (q, q') appears. We also compute other statistics of those sessions, such as, average session length, average number of clicks in the sessions, average position of the queries in the sessions, etc.
- **Time-related features.** We compute average time difference between q and q' in the sessions in which (q, q') appears, and the sum of reciprocals of time difference over all appearances of the pair (q, q').

The learning function. The next step for constructing the query-flow graph is to train a machine learning model to predict the label same_chain. The training dataset consists of approximately $5,000$ labeled examples; the labels were assigned by the authors of this paper.

We tested and compared many different machine learning approaches. As shown in Figure 1, the frequency of query pairs follows a power-law with a spike at 1. After experimenting with different settings, we decided to divide the classification problem into two subproblems, and thus the data were also partitioned into two training sets T_1 and T_2, by distinguishing between pairs of queries appearing together only once (we name this set T_1, which contain approximately 50% of the cases), and pairs appearing together more than once (we name this T_2). The distribution of the target variable same_chain is 66% positive and 34% negative in T_1, and 70% positive and 30% negative in T_2.

After various comparisons we selected the best models for T_1 and T_2 with respect to classification accuracy and simplicity of the model. For T_1 we adopted a very simple yet accurate *logistic regression* model using only 3 of the features available, namely (a) the Jaccard coefficient between sets of stemmed words, (b) the number of n-grams in common between the two queries, and (c) the time between the two queries in seconds. For T_2 instead we adopted a *rule-based model* consisting of a total of 8 simple rules (4 for each class).

We use the model we selected to assign the weight $w(q, q')$ to each edge (q, q'). In particular, we label each edge which has been classified as being in class 1 same_chain, with the conviction with which the model makes the prediction. All the edges that are classified in class 0, are labelled by 0, that corresponds to removing the edge from the query-flow graph G_{qf}.

The query-flow graph as a stochastic matrix. Next we consider normalizing the weights $w(q, q')$ of the query-flow graph so that the sum of the weights of the edges going out from each node is equal to 1. The result of such a normalization can be viewed as the transition matrix P of a Markov chain[1] whose states correspond to queries and where $P(q, q')$ is the probability that the query q' immediately follows the query q.

Adding the starting and terminal state. Finally, we have to describe how the two special nodes s and t enter in the query-flow graph $G_{\text{qf}} = (V, E, w)$. For each session $S \in \mathscr{S}(\mathscr{L}) = \{S_1, \ldots, S_m\}$, we add an edge (s, q) where q is the first query of the session S, and we assign it a weight of m'/m, where $m' \leq m$ is the number of times q appears at the beginning of a session. Similarly, let now q be the ending query of any session $S \in \mathscr{S}(\mathscr{L})$; moreover suppose that q appears c times across all the session, but only $m' \leq m$ times at the end of a session; then, we add an edge (q, t) with weight m'/c, and we multiply the weights of the other arcs of the form $(q, -)$ by $1 - m'/c$.

In Figure 2 we show a small snapshot of the query flow graph we produced. This contains the query "barcelona" and some of its followers up to a depth of 2, selected in decreasing order of count. Also the terminal node t is present in the figure. Note that the sum of outgoing edges from each node does not reach 1 just because not all outgoing edges (and relative destination nodes) are reported.

5. FINDING CHAINS

In this section we describe our first application of the query-flow graph: finding chains of queries in user sessions. As we have already mentioned, finding chains is a very im-

[1]The matrix itself is only substochastic, due to the presence of queries with no successors; when we need to turn it into a proper stochastic matrix (as we do for recommendation in Section 6), we add an artificial, uniformly weighted set of edges from every query q with no successors to all other queries.

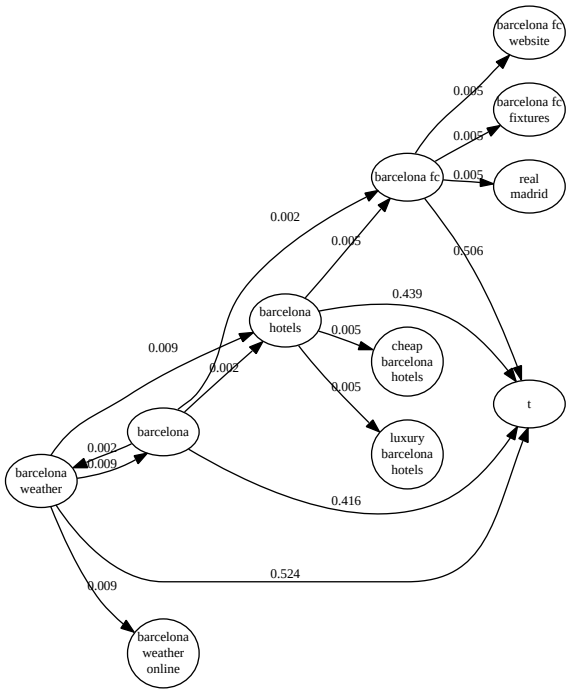

Figure 2: A portion of the query flow graph using the weighting scheme described on Section 4.

portant problem as it allows improving query-log analysis, user profiling, mining user behavior, and more.

The problem we consider is the following: We are given a supersession $S = \langle q_1, q_2, \ldots, q_k \rangle$ of one particular user. We are also given the query-flow graph, which has been computed with the sessions of S as part of its input. The chain-finding problem can also be defined in the case that the sessions of S have not participated in the construction of the query-flow graph. However, in this paper we focus on the former case and we leave the latter for future work.

One of the challenges of the problem we consider arises from our definition of chains: we allow chains not to be consecutive in the supersession S; in other words, the supersession S may contain many intertwined chains such as the ones shown in the Table 1. Previous work has mostly focused on the case where all chains are consecutive.

Chain #1	**Chain #2**
...	...
football results january 2nd	pointui forum
royal carribean cruises	audi ipswich
holidays	golfers elbow
motherwell football club	cox ipswich
...	...

Table 1: Two fragments from sessions containing non-consecutive chains.

The chain-finding problem can be formalized as follows: let us define a *chain cover* of $S = \langle q_1, q_2, \ldots q_k \rangle$ as a partition of the set $\{1, \ldots, k\}$ into subsets $C_1, \ldots, C_h$; each set $C_u = \{i_1^u < \cdots < i_{\ell_u}^u\}$ is thought of as a chain $C_u = \langle s, q_{i_1^u}, \ldots, q_{i_{\ell_u}^u}, t \rangle$, that is associated the probability

$$P(C_u) = P(s, q_{i_1^u}) P(q_{i_1^u}, q_{i_2^u}) \ldots P(q_{i_{\ell_u-1}^u}, q_{i_{\ell_u}^u}) P(q_{i_{\ell_u}^u}, t)$$

and we want to find a chain cover maximizing $P(C_1) \ldots P(C_h)$.

When a query appears more than once, "duplicate" nodes for that query are added to the formulation, which makes the description of the algorithm slighly more complicated than what is presented here. For simplicity of the presentation we omit the details related to queries appearing more than once below, which are not fundamental to the understanding of the algorithm.

TSP formulation. We shall now provide an alternative, equivalent formulation of the same problem. Given the sesson $S = \langle q_1, q_2, \ldots q_k \rangle$, consider a directed weighted graph $G_S = (V, E, \omega)$ with nodes $V = \{s, q_1, \ldots, q_k, t\}$, and arcs defined as follows:

- for every i and j with $i < j$, there is an arc (q_i, q_j) with
$$\omega(q_i, q_j) = \begin{cases} -\log P(q_i, q_j), \text{ if } i+1 < j \\ -\log \max\{P(q_i, q_j), P(q_i, t)P(s, q_j)\}, \text{ else;} \end{cases}$$
- for every i and j with $j < i$, there is an arc (q_i, q_j) with $\omega(q_i, q_j) = -\log P(q_i, t)P(s, q_j)$;
- for every i, there are arcs $(s, q_i), (q_i, t)$ with $\omega(s, q_i) = -\log P(s, q_i)$ and $\omega(q_i, t) = -\log P(q_i, t)$.

Intuitively, the graph G_S contains two kinds of arcs between the session queries:

- *green arcs*: a green arc from q_i to q_j is an arc whose weight is $-\log P(q_i, q_j)$, and it is present only if $i < j$; when we follow a green arc we are "going on" with a chain (possibly skipping some queries, those with indices $i+1, \ldots, j-1$, that will be part of some other chain);
- *red arcs*: a red arc from q_i to q_j is an arc whose weight is $-\log P(q_i, t)P(s, q_j)$, and it is present only if $i+1 \geq j$; when we follow a red arc we are deciding to stop the current chain; at this point, some other chain will start, and q_j is the starting point of the new chain. Notice that we shall always restart from the query q_j with smallest possible index that is as yet unassigned to any chain, so either $j < i$ or $j = i+1$.

We also have arcs starting from s and leading to t: these are used only for the very first query of the very first chain, say q_i, and for the very last query of the very last chain, say q_j, to account for the additional probability contribution given by $P(s, q_i)$ and $P(q_j, t)$, respectively.

We will prove that finding a chain cover maximizing $P(C_1) \ldots P(C_h)$ can be reduced to solving min-TSP on G_S. For sake of simplicity, assume for the moment that $P(q_i, q_{i+1}) = P(q_i, t)P(s, q_{i+1})$ for every $i < k$. Take a chain conver $C_1, \ldots, C_h$ (without loss of generality, let $i_1^1 < i_1^2 < \cdots < i_1^h$) and identify each C_u with a path $\hat{C}_u$ as follows:

$$\hat{C}_u \quad = \quad q_{i_1^u} \to \cdots \to q_{i_{\ell_u}^u}.$$

Because of the way we defined weights of green arcs, we have

$$\omega(\hat{C}_u) \quad = \quad -\log P(C_u) + \log A(s, q_{i_1^u}) A(q_{i_{\ell_u}^u}, t)$$

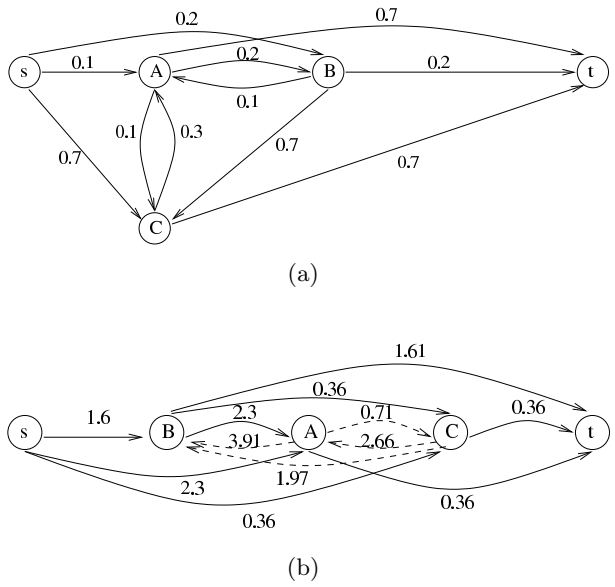

Figure 3: (a) A sub-graph from an example query graph, containing only three queries; (b) The graph G_S corresponding to a specific session $S = \langle B, A, C\rangle$.

Now, since $\omega(q_{i^u_{\ell_u}}, q_{i^{u+1}_1}) = -\log P(q_{i^u_{\ell_u}}, t) - \log P(s, q_{i^{u+1}_1})$, we have

$$\omega(s \to \hat{C}_1 \to \cdots \to \hat{C}_h \to t) = -log\left(P(C_1) \ldots P(C_h)\right),$$

and $s \to \hat{C}_1 \to \cdots \to \hat{C}_h \to t$ is a Hamiltonian path from s to t. Conversely, it is easy to see that every Hamiltonian path from s to t can be produced by a chain cover of S. So, finding a maximal chain cover can be reduced to finding a Hamiltonian path of minimum weight[2].

This is a version of the minimum travelling salesman problem (min-TSP), with asymmetric costs (the fact that the cycle is to contain the arc $t \to s$ does not make any difference, because all other arcs going out of t have infinite weight).

5.1 Example

In Figure 3, we show an example of a query-flow graph and the graph G_S corresponding to the session $S = \langle B, A, C\rangle$: logarithms have been approximated, and the Markov chain is shown only for the queries involved in the session. Dotted lines are used for red arcs. The solution to min-TSP is $s \to C \to B \to A \to t$ (with cost $0.36 + 1.97 + 2.3 + 0.36 = 4.99$), which corresponds to the cycle cover $C_1 = \langle s, C, t\rangle$ and $C_2 = \langle s, B, A, t\rangle$; note that $P(C_1) = 0.7 \cdot 0.7$ and $P(C_2) = 0.2 \cdot 0.1 \cdot 0.7$, with the overall product being $P(C_1) \cdot P(C_2) = .00686$ and $-\log(P(C_1)P(C_2)) \approx 4.99$ as expected.

[2]Dealing with the case $P(q_i, q_{i+1}) \neq P(q_i, t)P(s, q_{i+1})$ is easy, because then, even though there can be more than one cover associated with the same Hamiltonian path (one that breaks a chain between q_i and q_{i+1}, and one that does not), only one such chain will have maximal probability, and it is precisely the one that is used to assign weights in the graph.

5.2 Approximated greedy solution

It is well known that min-TSP is NP-hard even when weights are symmetric; exact branch-and-bound solutions exist, but are anyway rather slow and work reasonably only for few tens of nodes. Instead of trying to produce exact solutions, we content ourselves of a greedy heuristics that simply chooses every time the arc with minimum weight going out of the current node: in the following, we shall refer to this heuristic algorithm simply as the *ATSP algorithm*. The ATSP algorithm works in time $O(k^2)$, where k is the size of the supersession. It would be interesting to know how far the solution produced by this algorithm is from the exact solution on real data; on a more theoretical side, it would be nice to determine if our problem is still NP-hard, or if it is actually simpler, maybe polynomial. Both questions are left for future work.

5.3 Comparison with session timeout

In this section we describe our experiments for evaluating the chain-finding algorithm we propose, and compare it with a simple timeout-based method.

The query-flow graph is created as described in Section 4. For creating a training set for evaluating the session-breaking task, we sampled uniformly at random a set of 586 supersessions containing 2 queries or more—if there is only one query the task is trivial. Each of these 586 supersessions is classified by human editors using the following methodology: (*i*) first duplicate queries are eliminated, (*ii*) each query is assigned by the human editors to one chain (possibly non-consecutive), (*iii*) some queries remained unassigned in this process (due to the impossibility, by the human editor, to clearly map a query to one chain). The chains obtained in the above process consitute the "golden standard" with which we compare our algorithm.

We then apply the ATSP algorithm we described above for spliting the 586 supersessions into chains. For comparison we also implemented a "baseline" algorithm, which splits each supersession into sessions (using only the timeout threshold t_θ) and considers each resulting session as a chain.

Given a supersession S, the chains produced for S by the human evaluation or by the algorithms we test define a partition of S. We evaluate the ATSP and the Baseline algorithms by comparing the chains they produce with the chains produced by the human evaluation using the Rand index [17], a commonly employed measure of similarity between partitions. Notice that the chains produced by the human evaluation do not contain duplicate queries, while the chains produced by the ATSP and the Baseline algorithms might contain duplicates, so before computing the Rand index we remove duplicate queries.

Results. Given a supersession S, let $R_A(S)$ be the Rand index of comparing the chains produced for S by the ATSP algorithm with the "golden standard" chains for S, and let $R_B(S)$ be the Rand index of comparing the chains produced for S by the Baseline algorithm with the "golden standard" chains for S.

The distributions of R_A and R_B are summarized in Table 2.

From Table 2, it appears that the two algorithms are almost equivalent. However, taking a closer look at the results reveals that the seemingly similar performance is caused by many easy supersessions, e.g., superssesions consisting

Table 2: Rand index distributions for ATSP and Baseline.

	1st Qu.	Median	Mean	3rd Qu.
ATSP	0.7778	1.0000	0.8687	1.0000
Baseline	0.7521	1.0000	0.8554	1.0000

Table 3: Rand index distributions for ATSP and Baseline, when $R_B(S) < 1$.

	1st Qu.	Median	Mean	3rd Qu.
ATSP	0.5556	0.7778	0.7464	0.9022
Baseline	0.5100	0.7500	0.6984	0.8485

of one or two queries that the Baseline is able of handling correctly.

A more detailed analysis reveals that the ATSP algorithm is able of handling better more difficult supersessions. For instance, in the 92% of the cases in which $R_B(S) = 1$ we also have $R_A(S) = 1$. In the cases in which $R_B(S) < 1$ (supersession difficult for the Baseline) the average R_B score is 0.6984, while the average R_A score is 0.7464, a 6.4% improvement (see Table 3) On the other hand, in the cases in which $R_A(S) < 1$ (supersessions difficult for the ATSP) the average R_B score is 0.7248, while the average R_A score is 0.7140, which is only a 1.4% improvement of the Baseline with respect to the ATSP algorithm.

In other words, we can say that simple cases are treated comparatively well by the ATSP algorithm and the Baseline, while in difficult cases the ATSP algorithm clearly outperforms the Baseline; in Figure 4 we show the situation for the case $R_B(S) < 1$ through a scatter plot.

We note again that the ATSP algorithm has the ability to find intertwined chains, which, to our knowledge, is a significant novelty with respect to the current state of the art. We also note that given a supersession, the ATSP algorithm does not utilize at all the timestamp information of the queries, which, in fact, is the information exploited by the Baseline algorithm.

6. QUERY RECOMMENDATIONS

Most modern search engines include some form of automatic query recommendation, to suggest new queries that may be relevant to the current user's mission. Using query-log massive information to this purpose was suggested in [20]. Here we obtain query recommendations as an application of the query flow graph.

The query recommendation task is different from the session breaking task described in Section 5; while we can use the same query flow graph, we find that for the algorithm we propose it is more meaningful to use different weights on the edges. In particular, we define new weights $w'(q, q')$ as follows:

$$w'(q,q') = \begin{cases} \frac{\text{count}(q,q')}{\text{d}(q)} & \text{if } (w(q,q') > \theta) \vee (q = s) \vee (q = t) \\ 0 & \text{otherwise,} \end{cases}$$

where $\text{count}(q, q')$ is the number of times query q is followed by query q', or the number of times a chain starts with query q' if $q = s$, or the number of times a chain ends with query q if $q' = t$. The factor $\text{d}(q) = \sum_{q'} \text{count}(q, q')$ is used for normalization, and $\theta = 0.9$ is the value of confidence we require for the edge (q, q') to be composed of two queries in the same chain.

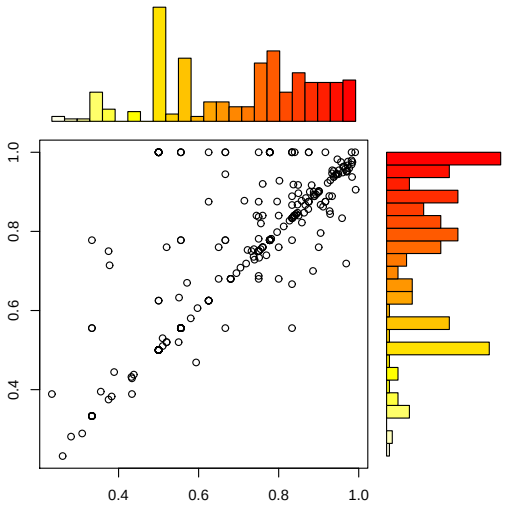

Figure 4: Every point in this plot corresponds to a supersession S with $R_B(S) < 1$; its coordinates are $(R_B(S), R_A(S))$. The fact that the points in the upper-left corner are denser than in the lower-right corner supports further the evidence that the ATSP algorithm outperforms the Baseline when $R_B(S) < 1$.

It is worth noting here, that intuitively the problem of query recommendation may benefit for handling query similarities in an non-symmetric way, and indeed, the query flow graph is strongly non-symmetric. Excluding the s and t nodes whose arcs are obviously not symmetric, 93% of the arcs in the graph do not have a reciprocal arc. Moreover, even for the few arcs that possess a reciprocal, the weights in both directions $w(q, q')$ and $w(q', q)$ are uncorrelated (Kendall's τ is about 0.26), and the same is true of w' (Kendall's τ is 0.16).

6.1 Recommendation by maximum weight

A simple recommendation scheme that uses the query flow graph is to pick, for an input query q, the node having the largest $w'(q, q')$. An example output from this scheme is shown on the first column of Table 4 for the queries "apple" and "jeep".

An issue with this method, that we observed for several test queries, is that it tends to "drift" towards those queries that are popular in the query log, but unrelated with the query at hand.

6.2 Recommendation by random walk

A recommendation algorithm can be built upon a measure of relative importance: when a user submits a query q to the engine, the recommendation that the engine provides should be the most important query q' relatively to q.

If we look at the problem under this point of view, we are naturally led to apply a form of personalized PageRank [9], where the preference vector is concentrated in a single node. Alternatively, this can be described as a random walk with restart to a single node [5]: a random surfer starts at the initial query q; then, at each step, with probability $\alpha < 1$

Max. weight	s_q	$\hat{s}_q$	$\bar{s}_q$
t	t	apple	apple
apple ipod	apple	apple fruit	apple ipod
apple store	apple ipod	apple ipod	apple trailers
apple trailers	apple store	apple belgium	apple store
amazon	apple trailers	eating apple	apple mac
apple mac	google	apple.nl	apple fruit
itunes	amazon	apple monitor	apple usa
pc world	argos	apple usa	apple ipod nano
argos	itunes	apple jobs	apple.com/ipod...
currys	pc world	apple movie ...	t
t	jeep	t	jeep
jeep cherokee	jeep trails	jeep	jeep cherokee
jeep grand ...	jeep kinderk...	jeep cherokee	jeep trails
jeep wrangler	jeep compass	jeep grand ...	jeep compass
land rover	jeep cherokee	bmw	jeep kinderkled...
landrover	swain and jon...	jeep wrangler	jeep grand ...
ebay	jeep bag	land rover	jeep wrangler
chrysler	country living ...	landrover	chryslar
bmw	buy range rov...	chrysler	jeepcj7
nissan	craviotto snare	google	buses to Knowl...

Table 4: Top 10 recommendation for the queries $q =$"`apple`", and $q =$"`jeep`" according to the baseline, and to the various random-walk scores proposed.

the surfer follows one of the outlinks from the current node chosen proportionally to the weights present on the arcs, and with probability $1 - \alpha$ (s)he instead jumps back to q.

This process describes the transition matrix A of a Markov chain that can be more formally defined as:

$$A = \alpha P + (1 - \alpha)\mathbf{1}\mathbf{e}_q^T$$

where P is the row-normalized weight matrix of the query flow graph, and $\mathbf{e}_j$ is the vector whose entries are all zeroes, except for the j-th whose value is 1.

Although A is not ergodic in general, as proven in [5] A is unichain as long as $\alpha \in [0\,..\,1)$, so it has a unique stationary distribution, namely, a unique distribution vector $\mathbf{v}$ such that $\mathbf{v}^T A = \mathbf{v}$. Such a distribution (called the *random-walk score relative to* q) can be computed using the power iteration method, and then employed to determine the relevance of all queries with respect to q, as explained below.

In all our experiments, we chose $\alpha = 0.85$, as it is customary in the PageRank literature [4], and used the ℓ_1-norm of the difference of two successive iterates to decide when to stop.

Recommendations can be deduced from the random-walk score by taking either the single top-scored query, or the best queries up to a certain lower score threshold. Notice that, in particular, if the most relevant query for q is t, this means that it is wise for the engine not to give any suggestion, because the query flow graph is showing that the chain at that point is more likely to end than to continue.

Using just the random-walk score, though, can be misleading, because in many cases a query has a high random-walk score simply because it is a very common query altogether; the situation, here, is not dissimilar to what happens in the classical weighting schemes used for document retrieval, like tf-idf, where the term frequency within a document needs to be discounted by the absolute importance of the term (the idf part of the formula).

Instead of using the pure random-walk score $s_q(q')$ of the query q' with respect to q, we can consider the ratio $\hat{s}_q(q') = s_q(q')/r(q')$ where $r(q')$ is the absolute random-walk score of q' (i.e., the one computed using a uniform preference vector). Experiments performed show that indeed in most cases $\hat{s}_q(q')$ produces rankings that are more reasonable, but sometimes tend to boost too much scores having a very low absolute score $r(q')$. To use a bigger denominator, we also tried with $\sqrt{r(q')}$ as $r(q') < 1$; this corresponds also to the geometric mean between $s_q(q')$ and $\hat{s}_q(q')$, that is

$$\bar{s}_q(q') = \sqrt{s_q(q') \cdot \hat{s}_q(q')} = \frac{s_q(q')}{\sqrt{r(q')}}.$$

Table 4 shows the output of the random-walk scoring and the adjusted variants discussed above: note that, except for the first few queries, the baseline soon "gets lost" in completely unrelated queries; s_q works well, but as expected popular queries (like "ebay") pollute the results; on the other hand $\hat{s}_q$ tends to overpenalize common queries, and tends to produce exotic recommendations ("apple belgium"), whereas $\bar{s}_q$ gives the most pertinent results.

6.3 Recommendation with history

A further step in the same direction is providing recommendation that depends not only on the last query input by the user, but on some of the last queries in the user's history. This approach may help to alleviate the data sparsity problem –the current query may be rare, but among the previous queries there might be queries for which we have enough information in the query flow graph. Basing the recommendation on the user's query history may also help to solve ambiguous queries, as we have more informative suggestions based on what the user is doing during the current session.

Using the same notation as before, suppose that $q_1, \ldots, q_k$ is the current query chain (ordered starting from the most recent); then, we consider the Markov process whose transition matrix is defined by

$$A = \alpha P + (1 - \alpha)\mathbf{1}\mathbf{e}_{q_1,\ldots,q_k}^T$$

where $\mathbf{v} = \mathbf{e}_{q_1,\ldots,q_k}$ is a vector whose entries are such that $v_{q_1} > v_{q_2} > \cdots > v_{q_k} > 0$. Equivalently, the overall process may be described using the random surfer metaphor, where $\mathbf{v}$ is the distribution used to choose the teleportation node, when teleportation is decided. Although other choices are possible, we always fixed $\mathbf{v}$ to be such that $v_q = 0$ for all $q \notin \{q_1, \ldots, q_k\}$, and $v_{q_i} \propto \beta^i$ for some $\beta < 1$.

Also in this case, we are not going to use the pure random-walk score $s_{q_1,\ldots,q_k}(q')$ of the query q' with respect to the sequence $q_1, \ldots, q_k$, but the adjusted score $\bar{s}_{q_1,\ldots,q_k}(q')$ instead. It is interesting to compare the relevance score $\bar{s}_{q_1,\ldots,q_k}(q')$ that can provide recommendation using the whole history with the score $\bar{s}_{q_1}(q')$ that can only exploit the last query.

Table 5 shows the output for two hypothetical chains. In the first one, the query $q' =$"`apple`"' is preceded by the query $q =$"`banana`"', or by the query $q =$"`beatles`"' ("Apple Records" is a record label founded by The Beatles). The parameter β is set to 0.8 and the scoring uses $\bar{s}_q$.

In Table 6, two actual query sessions are processed by the algorithm.

Table 5: Recommendations for the query q ="apple", considering that the previous query was "banana" (top) or "beatles"' (bottom).

banana → apple	banana
banana	banana
apple	eating bugs
usb no	banana holiday
banana cs	opening a banana
giant chocolate bar	banana shoe
where is the seed in anut	fruit banana
banana shoe	recipe 22 feb 08
fruit banana	banana jules oliver
banana cloths	banana cs
eating bugs	banana cloths

beatles → apple	beatles
beatles	beatles
apple	scarring
apple ipod	paul mcartney
scarring	yarns from ireland
srg peppers artwork	statutory instrument A55
ill get you	silver beatles tribute band
bashles	beatles mp3
dundee folk songs	GHOST'S
the beatles love album	ill get you
place lyrics beatles	fugees triger finger remix

Table 6: Recommendations for two actual query chains.

music	facebook → gabriella → music
music	music
yahoo music	gabriella
music videos	yahoo music
music downloads	music videos
free music	music downloads
yahoo music videos	free music
music yahoo	gabriella sweet like me
free music videos	lighting bug rotherham
yahoo music launch	ccp npa ndf
free music downloads	gabriela lighting

evening dress	orion → orion dress orion evening dress → evening dress
evening dress	evening dress
formal evening dress	orion evening dress
red evening dress	formal evening dress
myevening dress	red evening dress
prom 008 dresses	long dressess
long dressess	myevening dress
evening dress uk	fashion women dress
fashion women dress	prom 008 dresses
dresses for the evening	evening dress uk
1900evening dress	1900evening dress

7. CONCLUSIONS

The query-flow graph summarizes a query log in a compact representation. This representation can be obtained efficiently from the source data and enables several key search and mining operations. The query-flow graph is sparse, and about half of the query pairs appear only once in the query log. Also, the graph is strongly non-symmetrical, as 93% of the edges have no reciprocal edge.

In this paper, we have shown two key applications in usage mining that are supported by the query-flow graph. We have shown a method that exploits the information in the query-flow graph for segmenting the user sessions into logically-coherent query chains. We have also shown several methods for generating query suggestions based on random walks in the query-flow graph.

Extensive evaluation and tuning of these methods is necessary to implement them effectively in practice. So far we have shown that these tasks can be implemented efficiently using the abstraction we have developed here. Specific aspects to look at in future work include: features for the query segmentation model, weighting schemes for the recommendation systems, scoring methods for the output of the random walks, and better evaluation methods.

8. REFERENCES

[1] R. Baeza-Yates. Graphs from search engine queries. In *Theory and Practice of Computer Science (SOFSEM)*, volume 4362 of *LNCS*, pages 1–8, Harrachov, Czech Republic, January 2007. Springer.

[2] R. Baeza-Yates and A. Tiberi. Extracting semantic relations from query logs. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 76–85, New York, NY, USA, 2007. ACM Press.

[3] R. A. Baeza-Yates, C. A. Hurtado, and M. Mendoza. Query recommendation using query logs in search engines. In *EDBT Workshops*, volume 3268 of *LNCS*, pages 588–596. Springer, 2004.

[4] M. Bianchini, M. Gori, and F. Scarselli. Inside pagerank. *ACM Trans. Interet Technol.*, 5(1):92–128, 2005.

[5] P. Boldi, V. Lonati, M. Santini, and S. Vigna. Graph fibrations, graph isomorphism, and PageRank. *RAIRO Inform. Théor.*, 40:227–253, 2006.

[6] L. Catledge and J. Pitkow. Characterizing browsing behaviors on the world wide web. *Computer Networks and ISDN Systems*, 6(27), 1995.

[7] K. Collins-Thompson and J. Callan. Query expansion using random walk models. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 704–711, New York, NY, USA, 2005. ACM.

[8] N. Craswell and M. Szummer. Random walks on the click graph. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 239–246, New York, NY, USA, 2007. ACM Press.

[9] K. Csalogány, D. Fogaras, B. Rácz, and T. Sarlós. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Math.*, 2(3):333–358, 2005.

[10] B. M. Fonseca, P. B. Golgher, E. S. de Moura, and N. Ziviani. Using association rules to discover search

engines related queries. In *LA-WEB '03: Proceedings of the First Latin American Web Congress*, Washington, DC, USA, 2003. IEEE Computer Society.

[11] D. He and A. Göker. Detecting session boundaries from web user logs. In *Proceedings of the BCS-IRSG 22nd annual colloquium on information retrieval research*, pages 57–66, Cambridge, UK, 2000.

[12] D. He, A. Göker, and D. J. Harper. Combining evidence for automatic web session identification. *Inf. Process. Manage.*, 38(5):727–742, September 2002.

[13] R. Jones and F. Diaz. Temporal profiles of queries. *ACM Trans. Inf. Syst.*, 25(3), July 2007.

[14] R. Jones and K. L. Klinkner. Beyond the session timeout: automatic hierarchical segmentation of search topics in query logs. Submitted for publication, 2008.

[15] B. Piwowarski and H. Zaragoza. Predictive user click models based on click-through history. In *CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 175–182, New York, NY, USA, 2007. ACM.

[16] F. Radlinski and T. Joachims. Query chains: learning to rank from implicit feedback. In *KDD '05: Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 239–248, New York, NY, USA, 2005. ACM Press.

[17] W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66:622–626, 1971.

[18] J. Teevan, E. Adar, R. Jones, and M. A. S. Potts. Information re-retrieval: repeat queries in yahoo's logs. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 151–158, New York, NY, USA, 2007. ACM.

[19] J.-R. Wen, J.-Y. Nie, and H.-J. Zhang. Clustering user queries of a search engine. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 162–168, New York, NY, USA, 2001. ACM.

[20] Z. Zhang and O. Nasraoui. Mining search engine query logs for query recommendation. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 1039–1040, New York, NY, USA, 2006. ACM.