Clustering Query Refinements by User Intent

Eldar Sadikov
Stanford University
eldar@cs.stanford.edu

Jayant Madhavan
Google Inc.
jayant@google.com

Lu Wang
Google Inc.
luwang@google.com

Alon Halevy
Google Inc.
halevy@google.com

ABSTRACT

We address the problem of clustering the refinements of a user search query. The clusters computed by our proposed algorithm can be used to improve the selection and placement of the query suggestions proposed by a search engine, and can also serve to summarize the different aspects of information relevant to the original user query. Our algorithm clusters refinements based on their likely underlying user intents by combining document click and session co-occurrence information. At its core, our algorithm operates by performing multiple random walks on a Markov graph that approximates user search behavior. A user study performed on top search engine queries shows that our clusters are rated better than corresponding clusters computed using approaches that use only document click or only sessions co-occurrence information.

1. INTRODUCTION

Web search engines today often complement the search results with a list of related search queries. The related searches are either presented at the top or bottom of the search results page (for Google and Yahoo!) or as a navigation bar on the left (for Bing). For example, given the query mars, Google.com returns the related queries mars god of war, mars planet, venus, jupiter, etc. These related searches help users to find and explore information related to the original query. Furthermore, because users often provide short queries with little or no context, related queries allow users to specify their information needs [2]. For example, by clicking on mars god of war, a user signals interest in the Roman god as opposed to the planet Mars.

Related queries are typically mined from the query logs by finding other queries that co-occur in sessions with the original query [16]. Specifically, *query refinements*, a particular kind of related queries, are obtained by finding queries that are most likely to follow the original query in a user session. For many popular queries, there may be hundreds of related queries mined from the logs. However, given the limited available space on a search results page, search engines typically only choose to display 5-10 related queries.

We address the problem of clustering query refinements. Specifically, our goal is to group the refinements into clusters that are likely to represent distinct information needs. For example, for mars, we would like to separate out queries that pertain to the Roman god Mars, from those that pertain to facts about the planet Mars, from those that represent other planets in the Solar System in general, etc.

There are multiple motivations for clustering query refinements and related queries in general. First, having space for only a few related queries in the results page, it is critical to select a diverse set that corresponds to different information needs. We do not want all the related queries for mars to be about the planet alone, but rather be representative of the various interests people might have. However, current solutions to selecting related queries rely more on frequency than on diversity. For example, while the results of our clustering for mars indicate that the second most popular cluster pertains to the Mars chocolate bar, none of the queries in this cluster, e.g., mars candy or mars chocolate bar, individually appear in the top 10 most frequent refinements for mars. As a result, this user intent of mars is not represented on the search results page.

A second motivation is to use clustering to improve the placement of related queries on the search results page. Related queries are often placed in rows or columns. If these columns correspond to cluster groups, they are potentially easier to understand.

The third motivation is to improve related-query suggestions *across* user sessions. For example, if a user poses the query pluto after mars, it is more likely that their interest is the Solar System rather than the Disney character Pluto the Dog. Hence, it makes more sense to propose related searches for pluto that pertain to planets or facts about planets, rather than Disney characters.

Finally, the clusters provide a summary of all the possible diverse interests and information needs that people have about a given query (as expressed by the queries they pose). For example, our results indicate that for mars, there are distinct clusters of refinements about planets in the Solar System, Mars the Roman god, the Mars candy bar, and a Japanese comic strip. About the planet itself, there are distinct clusters for facts about the planet, the rovers sent by NASA, and speculation about life and water on the planet. Such summaries of the information relevant to a query can form the basis for other search-result interfaces, such as mashups that provide topic summaries (e.g., [18]).

One approach to clustering refinements would be to group them based on their respective search results, e.g., grouping queries that shared many similar clicked documents [5, 26], or based on the similarity in the vocabulary of the clicked documents [3]. However, these techniques fail to achieve all our clustering goals. For example, for mars, the two possible

WWW 2010, April 26–30, 2010, Raleigh, North Carolina, USA.
ACM 978-1-60558-799-8/10/04.

refinements venus and jupiter correspond to the same user intent of researching planets. However, the two queries are unlikely to retrieve any search results in common, let alone document clicks, and hence will not be clustered together.

A second approach would be to group refinements based on their occurrences within user search sessions, e.g., grouping queries that co-occur with similar sets of other queries [13]. This approach turns out to be effective for clustering queries that are unrelated content-wise, e.g., venus and jupiter. However, there are two challenges with the session data. First, session co-occurrences can be sparse (in comparison to document clicks), especially for less frequent queries, making it hard to infer statistically significant relationships. Second, there is a often "drift" in user intent within the same session. For example, mars might be followed by neptune and pluto, then onto pluto pictures and then pluto the dog, making many more refinements be transitively related to each other.

We describe an approach that combines document-click analysis and session analysis. We model user behavior as a graph whose nodes are refinements and clicked documents, such that clusters of nodes in the graph correspond to different user intentions. The graph has a natural interpretation as a Markov model, and we can characterize each refinement-node in the graph by its distribution vector on the absorbing states of the model. This interpretation also provides a computational benefit, because we obtain a polynomial-time clustering algorithm (over the vectors) versus an NP-hard graph-clustering problem. We describe a set of experiments that show that users strongly prefer the clusters produced by our method and provide analysis of the resulting clusters.

2. PROBLEM FORMULATION

We begin by introducing our terminology and formulating our clustering problem. In our discussion we assume that we have access to (completely anonymized) search-query log, and that the queries are divided into *sessions*. A session is a sequence of queries posed by a single user within a short period of time (10 minutes in our case). Our goal is to cluster query *refinements*.

DEFINITION 1. *A query r is said to be a* refinement *of a query q, if r follows q in a session. We denote by $R(q)$ all the refinements of a query q.* □

Note that q does not need to be the first query in a session in order for r to be a refinement. Furthermore, the model we describe is agnostic to how query refinements are collected, but we provide Definition 1 because it is the most common method for determining refinements. We also model co-occurrence of query refinements:

DEFINITION 2. *The set of co-occurring queries of q_i, denoted by $Q(q_i)$, is the set of queries q_j, such that q_i and q_j occur in the same session.* □

We assume that the query log records the documents clicked in response to a query.

DEFINITION 3. *The document set of a query q_i, denoted $D(q_i)$, is the set of documents that users click on after being presented the answers to q_i.* □

Modeling user behavior as a graph: Given the query logs and a query q, our goal is to cluster the query refinements of q into a set of different information needs. The intuition underlying our clustering algorithm is that that two refinements are similar if they lead to the same content within a typical search session. Our graph model is based on the following typical user behavior.

Consider a user who might start a search session with an underlying intent, i.e., some, possibly abstract, information need. She poses a query she believes will satisfy her information need, e.g., mars. If none of the search results satisfy her, she might pose another query that is likely to better capture her specific need, e.g., mars pictures. If a result does satisfy her, she will inspect the results further by clicking on one or more of them. After clicking on a result, she might also pose another query to find more results relevant to her information need, e.g., venus after mars, when the underlying intent is to research planets.

We construct a graph, $G(q) = (V, E)$, that captures this behavior as follows (see Figure 1(a) for an example). There is a node in V for the query q, for each of the query refinements in $R(q)$, and for any document that is clicked, i.e., every element of $\cup_{r \in R(q)} D(r)$. The set of edges, E, includes:

- for every $r \in R(q)$, (q, r), i.e., edges from the query q to each of its refinements;
- for every $r \in R(q)$, and every $d \in D(r)$, (r, d), i.e., edges from r to each of its clicked documents, and
- for every $r \in R(q)$ and $r_i \in Q(r) \cap R(q)$, (r, r_i), i.e., edges connecting co-occurring refinements.

Our discussion is typically in the context of a given query q, so we refer to the graph simply as G rather than $G(q)$.

We assign weights to edges in G from information available in the query logs. For edges of the form (r, d) we assign weight $w(r, d)$ proportional to the probability of clicking on d as a search result of r. For edges of the form (r, r_i), where r is either the query q or one of its refinements, we assign weight $w(r, r_i)$ proportional to the probability of r and r_i co-occurring in a search session. Note that we do not consider probabilities conditioned on sessions starting with the query q. This is because it is unlikely that such conditioned probabilities can be obtained reliably, given the sparsity of such information in the query logs.

Note that we do not have edges from documents to queries. This is not a limitation, because if a user were to proceed as follows: $q \to r_1 \to d \to r_2 \ldots$, then r_2 is also, by definition, a refinement of q, and all subsequent interactions after r_2 will be accounted in G.

Clustering nodes in the graph: The intuition underlying our clustering technique is the following. Two query refinements of q, r_i and r_j, represent the same underlying intent if a user typically reaches the same documents in sessions where q is followed by r_i and in sessions where q is followed by r_j.

Consider the following example. When a user poses the query mars, with the intent of researching planets, it is likely that she then subsequently queries one or more of venus, earth, jupiter, etc. In each case, she might click on the Wikipedia document about the corresponding planet. Other users with the same intent are likely to pose a subset of the same queries (albeit in different orders) and click on the same documents during their search sessions. In contrast, if the user intent was the Mars candy bar, subsequent refinements are likely to be mars candy, mars chocolate, etc., with the document clicks in the corresponding sessions leading to very different documents.

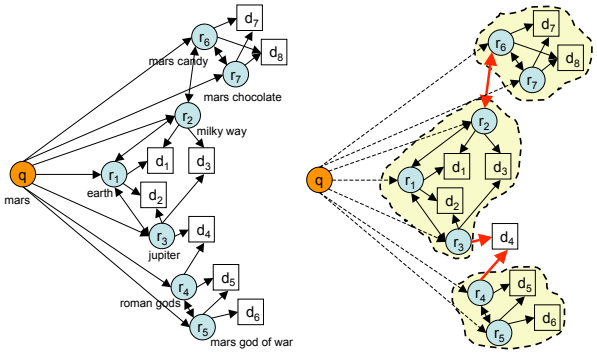

Figure 1: (a) Graph G models user search behavior. (b) Partition of its refinement nodes into 3 clusters so as to minimize the cost function in Problem 1.

There are two important points to note that distinguish our clustering technique from previous work on query clustering [5]. First, we consider *entire paths* from the query to the set of documents, rather than just the documents viewed upon seeing the query answers. This is important because the path that the user follows provides additional context about their intent, and for certain queries, the relevant documents are only found after some exploration. Second, we consider the paths followed from the refinement r_i to r_j in the *context* of posing the original query q by restricting our graph to include only the refinements of q (rather than all user queries). For example, the queries sun and pluto, though unambiguous in the context of mars, might lead to a different set of pages (e.g., pages about the company Sun Microsystems and the Disney character Pluto the Dog, respectively) if they are not in sessions with mars.

We capture the above intuition by hypothesizing that the set of documents reachable from a query refinement r_i, after starting from the query q, are representative of the user's underlying *intent* in selecting r_i after q. We can now formulate our problem as clustering by intent. Specifically, if there is an edge (r_i, r_j) in the graph, or both r_i and r_j have edges to the same documents, then users in r_i's typical session will reach the same documents as users in r_j's typical session. On the other hand, if there is no path from r_i to r_j, and they have few (or no) common documents, then users in r_i's typical session are unlikely to visit the same documents as users in r_j's typical session. Accordingly, if we can cluster the queries such that we minimize (1) the number of edges between query nodes in different clusters and (2) the document nodes that have edges from multiple clusters, then, we effectively satisfy the proposed intuition.

Figure 1(b) illustrates a partition of the query nodes into 3 clusters. There is only one edge (r_2, r_6) between refinements in different clusters, and only one document d_4 with edges from queries in different clusters. The (q, r_i) edges are excluded since we are only interested in clustering refinements.

Our clustering based on partitioning the graph G naturally maps to a variant of the classical *min k-cut* problem [14]. Specifically, suppose we know the number of desirable clusters k. We can define our clustering objective as follows:

PROBLEM STATEMENT 1. *Given a graph, G, and the number, k, of required clusters, partition the set of refinement vertices in G into proper subsets $\mathcal{R} = \{R_1, \ldots, R_k\}$, such that the following cost function is minimized:*

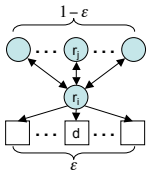

Figure 2: Normalizing transition probabilities

$$\sum_{(r_i,r_j)} w(r_i, r_j) \times \mathbf{1}\{r_i \in R_l, r_j \in R_m, R_l \neq R_m\} + \sum_{(r_i,d)} w(r_i, d) \times \mathbf{1}\{r_i \in R_l, \exists (r_j, d) \in E, r_j \in R_m, R_l \neq R_m\}$$

where $\mathbf{1}\{c\}$ is an indicator variable equal to 1 if condition c is true and 0 otherwise.

Unlike the classical min k-cut, our goal is to cluster only the nodes corresponding to query refinements, whereas the document nodes are not included in the clusters. Using a simple reduction from the original min k-cut problem, we can prove that the problem of clustering refinements is NP-hard for an arbitrary k:

PROPOSITION 1. *The problem of clustering the refinements of a query, as defined in Problem 1, is NP-hard.* □

Although an $O(n^{k^2})$ algorithm exists for the min k-cut (for a fixed k) [14], it is not practical for any reasonable k. There also exist approximation algorithms [21]. In the next section we describe a more efficient clustering algorithm that is based on interpreting our graph as a Markov model.

3. MARKOV MODEL INTERPRETATION

The key insight underlying our clustering technique is that the graph of transitions, G, has a very natural interpretation as a Markov model, describing transition probabilities between states. Furthermore, the absorbing states of this Markov model are the nodes in G corresponding to clicked documents. As a result, we can characterize each of the query-refinement nodes in G by the vector of probabilities of reaching each of the absorbing states. Clustering the query-refinement nodes based on these vectors is consistent with our intuition of clustering by user intent. Furthermore, as an added benefit, the computational complexity of clustering is significantly lower than that of min k-cut.

Section 3.1 explains how we transform G into a Markov model; Section 3.2 describes the important properties of the model; Section 3.3 discusses how our algorithm handles cases in which users drift to other topics during a session; and Section 3.4 describes the clustering algorithm.

3.1 Creating a Markov model

Our construction of the graph G naturally lends itself to a Markov process interpretation. The weights on the edges of the graph were computed based on the probability of transitions between the states. Hence, we can view each node as a Markov state with the edge weights being the transition probabilities between the corresponding states.

Transition Probabilities: To ensure that the graph represents a valid Markov chain, we normalize the outgoing edge weights from each node to sum up to 1. We do so by defining a parameter ϵ, the *document escape probability*, which represents the probability that there will be a transition from a query-refinement node to a document node. Consequently, with probability $1-\epsilon$, there will be transition

from a refinement node to another refinement node (see Figure 2). Although the value of ϵ does not significantly affect the results, we will soon see that ϵ is a useful parameter in our model.

We define the transition probability matrix, P, for the graph G, as follows:

- for each (r_i, d), where $d \in D(r_i)$, and $n_d(d|r_i)$ is the number of times a user clicks on the document d, a result of the query r_i,
$$P[r_i, d] = \epsilon \times \frac{n_d(d|r_i)}{\sum_{d_k \in D(r_i)} n_d(d_k|r_i)}$$
- for each (r_i, r_j), where r_i and r_j are both refinements of q and $r_j \in Q(r_i)$ (i.e., $r_j \in R(q) \cap Q(r_i)$), and $n_s(r_i, r_j)$ is the number of sessions in which r_i and r_j co-occur,
$$P[r_i, r_j] = (1 - \epsilon) \times \frac{n_s(r_i, r_j)}{\sum_{r_k \in R(q) \cap Q(r_i)} n_s(r_i, r_k)}$$
- for each document d (all of which are terminal in G), we add self-transitions
$$P[d, d] = 1$$

Note that due to the sparsity of session logs, we do not restrict $n_s(r_i, r_j)$ to only the sessions where r_j follows r_i, and instead consider all sessions in which they co-occur. Although it makes edges bidirectional, the weights on edges are not symmetric (due to the denominator), thus if there is a strong directionality to some edges, it is still preserved. Also, as mentioned in Section 2, we restrict the transitions between refinement nodes to be only those in the context of the original query q, i.e., its set of refinements R. We consider transitions to nodes not in R in Section 3.3.

Though the construction of the Markov model is dependent on the original query q, we do not consider q and its transitions to be a part of the model. This is because, as we shall soon find out, the transitions from q have no bearing on our algorithm.

Markov Chain Properties: Recall from the Markov process theory [24], that a state is *absorbing*, if it is impossible to escape from it, and *transient*, if there is a non-zero probability of never returning to it. Using this terminology, each refinement node is a transient state, while each document node is an absorbing state (only self-transitions). Moreover, since at least one absorbing state is accessible from each of the states, the Markov chain is said to be *absorbing*. In other words, if one were to perform an infinite-step random walk on this Markov chain (starting at any state), one will always escape the refinement states and be absorbed by one of the document states.

It follows from Taylor and Karlin [24] (p. 169), that in our context, the probability of absorption in any particular absorbing state k depends on the initial state. As seen in Figure 3, if a random walker starts at r_3, she is likely to satisfy her intent at the documents close to r_3, i.e. d_4, d_2, d_1, d_3. On the other hand, if she starts at r_7, she is likely to satisfy her intent at d_8, d_7, d_3, d_1.

Clustering by Absorption Distributions: We can use the fact that the absorption distribution is conditioned on the start node to determine which documents are most descriptive of a query. Specifically, we perform a random walk starting from each of the refinements r_i of the query q, and obtain their specific absorption distribution vector $\vec{l_i}$. Each entry in $\vec{l_i}$ will correspond to a document node and equal

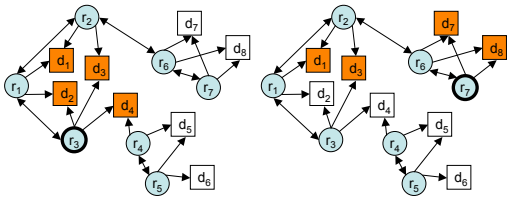

Figure 3: 3-Step random walk that originates from r_3 in (a) and r_7 in (b). Absorbing document nodes are highlighted.

the probability of reaching that document at the end of an infinite random walk starting from r_i. Then, to measure similarity between two refinements r_i and r_j, we compare their corresponding absorption distribution vectors $\vec{l_i}$ and $\vec{l_j}$. This allows us to cluster refinements as points in some n-dimensional space and find which refinements are likely to represent the same user intent.

The pseudo-code of the algorithm is shown in Algorithm 1. The inputs are the graph $G(q)$ constructed as described earlier and the number of desired clusters k. There are two parameters: ϵ, the document escape probability, and n, a parameter to the random walk which we describe later. In the first step, we initialize the transition matrix as described in this section. We describe each of the other steps of the algorithm in the next subsections.

Algorithm 1 `clusterRefinementsByIntent`$(G(q), k, \epsilon, n)$

$P \leftarrow$ `initializeTransitionMatrix`$(G(q), \epsilon)$
$P' \leftarrow$ `calculateLimitingDistributions`(P, n)
$L \leftarrow$ `extractAbsorptionDistributions`$(G(q), P')$
$\mathcal{R} \leftarrow$ `clusterVectors`$(G(q), L, k)$
return $\mathcal{R}$

Observe that the similarity of refinements that share clicked documents depends on the probability mass flowing from them to the documents. On the other hand, the similarity of refinements that are connected by edges is proportional to the weights of their connecting edges. For example, in Figure 3, if the transition probability of (r_2, r_6) is w, then at least $w \times epsilon$ of the probability mass originating at r_2 is absorbed by d_7 and d_8 (in contrast to the entire mass flowing directly from r_6 to d_7 and d_8). Thus, for large w, r_2 and r_6 are more similar. Thus, by pushing the probability mass down the edges with more weight, our approach is able to cluster together refinements connected by high-weight edges. This way, edges with high weight end up inside of the clusters and edges with low weight – between the clusters, thus minimizing the cost function in Problem Statement 1.

3.2 Computing the distributions

We now describe how we compute the absorption distributions of our Markov model. Given the transition matrix P, we note that the matrix product $P \times P$ is such that its $[i, j]$ entry will be the 2-step transition probability from state i to state j. Following the same pattern, $P^n[i, j]$ has the n-step transition probability from i to j. As $n \to \infty$, P^n approaches the limiting distribution. In particular, $\lim_{n\to\infty} P^n[i, j]$ entry is equal to the limiting fraction of time random walker spends in state j when starting from state i [24].

Since our Markov chain is absorbing, we can make a stronger claim. Specifically, the *visit probability* (also known as *hitting probability*), i.e., the probability of visiting an *absorbing* state d within an n-step random walk starting at some state

r_i, is equal to the probability of transitioning from state r_i to state d in n steps. Hence, we can state the following:

PROPOSITION 2. *Given the transition matrix P computed for our Markov model, the row of $\lim_{n\to\infty} P^n$ corresponding to a node v is the visit probability distribution vector of the random walk started at v over the absorbing states.* □

The visit probability distribution for a refinement r_i captures the probability that a user will eventually reach different documents during a search session that starts with q and includes r_i. It is thus representative of our hypothesized user intent underlying r_i in the context of q.

Using Proposition 2, the method `calculateLimitingDistributions(P,n)` in Algorithm 1 is then a simple matrix product P^n that approximates the visit probability distribution for a suitably chosen n (discussed later in this section).

Note here that calculating P^n is not the only way to calculate visit (or equivalently, absorption) probabilities. An alternative method is to use the *fundamental matrix* of our Markov chain [24]. Although this method is of the same computational complexity as the one we use ($O(n^3)$ for an $n \times n$ matrix), it requires finding inverse of a matrix and, thus, is slightly less intuitive. More importantly, for realistic values of ϵ, we show that our method converges very quickly, making the matrix-product method even more appealing.

Extracting Absorption Distributions: In any long random walk on an absorbing Markov model all the probability mass gets absorbed by the absorbing states. This implies that the columns of $\lim_{n\to\infty} P^n$ corresponding to the transient states (i.e., refinements in our case) will always be zero. Hence, when working with limiting distributions, we only need to consider the columns corresponding to the document states (i.e. absorbing states). Using this fact and Proposition 2, we show in Algorithm 2 how we extract absorption distributions from P^n.

Algorithm 2 `extractAbsorptionDistributions(G,P')`

```
for r_i ∈ R do
    l_i ← vector of size |⋃_{r_j∈R(q)} D(r_j)|
    for d ∈ ⋃_{r_j∈R(q)} D(r_j) do
        l_i[d] = P'[r_i, d]
    end for
end for
L = {l_1, ..., l_r}, where r = |R(q)|
return L
```

Parameters ϵ and n: Recall that the parameter ϵ in our Markov model controls how likely a user is to click on a document from any refinement node. In a sense, this parameter controls how "exploratory" we believe the user is, and in practice it controls the convergence rate of our method. Our experiments showed that for any practical values of ϵ, the algorithm converges quickly. For example, with a value of $\epsilon = 0.6$, after only four iterations (i.e., transition matrix multiplications), the unabsorbed probability mass remaining was only $(0.4)^4 = 0.0256$. Even for $\epsilon = 0.3$ (corresponding to a very "exploratory" browsing behavior), it takes just 7 iterations to absorb over 90% of the probability mass. Since the number of clustered refinements almost never exceeds $1,000$, it is possible to calculate P^n quickly in practice. Although changing ϵ and number of iterations does not fundamentally change the results, we found lower number of iterations (3-5) and higher values of ϵ (0.5-0.7) to work better, thus suggesting that most queries have rather focused user intents.

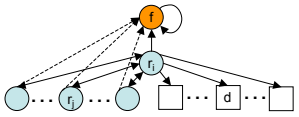

Figure 4: From each related query, we add a transition to the absorbing off-topic state f

3.3 Off-Topic Drift

For ease of exposition, when describing the model, we did not talk about the possibility of a *drift* in user intent to another topic. We have been assuming so far that after issuing original query q, the user is always going to satisfy her current intent by submitting one or more query refinements, and eventually clicking on one of their documents. Realistically, however, it is possible that before clicking on any URL, the user may change her mind and submit a query that would *not* be among the query refinements of the original query q. In other words, the user may abandon whatever intent or information need she is after and pursue a new information need. In this section we show how this off-topic drift could be modeled in our framework.

Specifically, we are going to add to our Markov model a new state f that will signify the user's transition off-topic (see Figure 4). Then, from each of the query refinements $r_i \in R(q)$, we add an off-topic transition (r_i, f). Once the user switches off-topic, we assume she does not come back to the original topic, thus we make f an absorbing state with no outgoing edges (only a self-transition with probability 1).

To complete our construction, we need to determine the transition probability for (r_i, f). Our original model without the off-topic drift is essentially equivalent to that probability being 0. One option would be to set the probability to some constant, and then respectively normalize other transition probabilities. However, using a constant seems inaccurate. Consider the refinements mercury and water on mars for the query mars. Clearly, mercury is more likely to take us off-topic, since it has a number of interpretations, such as the car make and the chemical element, that are unrelated to the original query mars. Hence, it seems natural to have the transition probability for (r_i, f) be dependent on r_i.

Recall from our construction of G, that for a given refinement r_i of q we look at transitions in *all* sessions (whether they begin with q or not), but only consider the transitions to other refinements of q. To estimate the drift beginning from r_i we can now look at the transitions from r_i to queries that are not in $R(q)$. If we sum over all such queries q', we can effectively gage related query's off-topic drift.

Formally, for a query refinement $r_i \in R(q)$, we are going to set (r_i, f) transition probability as:

$$P[r_i, f] = (1-\epsilon) \times \frac{\sum_{q' \in (Q(r_i) - R(q))} n_s(r_i, q')}{\sum_{q' \in Q(r_i)} n_s(r_i, q')}$$

Accordingly, all transitions (r_i, r_j) between any two query refinements $r_i, r_j \in R, r_i \neq r_j$ will no longer be conditioned on r_j being in R:

$$P[r_i, r_j] = (1-\epsilon) \times \frac{n_s(r_i, r_j)}{\sum_{q' \in Q(r_i)} n_s(r_i, q')}$$

The varying off-topic drift plays an interesting role in our clustering. If we leave it out of the model, then the entire transition probability mass of r is pushed by the random walk to other nodes that do not quite merit it. For example, consider the case where r_i has a high off-topic probability, and only one on-topic transition to a different refinement r_j.

Ignoring off-topic transitions implies that we are saying that all the probability mass is transferred from r_i to r_j. This will make the absorption distribution of r_i and r_j almost identical, deeming them to be more similar than they really are. This effect can be exacerbated by the probabilities being pushed transitively along the Markov chain to other refinement nodes, thereby rendering the clustering ineffective.

To account for the fact that transition probabilities are not conditioned on q, we could also decrease slightly the off-topic probability summed over all $q' \in (Q_i - R)$ (we would need to adjust transitions between queries in R accordingly as well). This is effectively the middle ground between the original model with no off-topic probability and the proposed model where the off-topic probability sum is taken as is. Although we have not experimented with this much, this could be an interesting future direction to pursue.

3.4 Clustering

Thus far we described how to map the problem of clustering query refinements into a problem of Euclidean-vector clustering. We now describe how we implemented the clustering step.

Algorithm 3 `clusterVectors`(G, L, k)

$\mathcal{R} \leftarrow \emptyset$
for $r_i \in R$ **do**
  $\mathcal{R} \leftarrow \mathcal{R} \cup \{r_i\}$
end for
while $(|\mathcal{R}| > k)$ **and**
    $(\max_{R_l \neq R_m \in \mathcal{R}} \mathit{completelink}(R_l, R_m) > 0)$ **do**
  $\langle R_l, R_m \rangle \leftarrow \arg\max_{R_l \neq R_m \in \mathcal{R}} \mathit{completelink}(R_l, R_m)$
  $R_l \leftarrow R_l \cup R_m$; $\mathcal{R} \leftarrow \mathcal{R} - R_m$
end while
return $\mathcal{R}$

Clustering Algorithm: One of the advantages of our model is that it can employ any algorithm for clustering Euclidean vectors (e.g., hierarchical, density based, partitional, graph based) [17, 19].

However, methods that suffer from chaining or are designed for elongated transitive clusters won't perform very well in our context [19]. As mentioned before, user intents tend to drift within sessions, thus many queries may be transitively (via 2-4 other queries) related to almost every other query. For this reason, single-link and group-average clustering algorithms perform poorly here. On the other hand, complete-link clustering is very effective.

We experimented with different clustering algorithms and similarity measures, and we found complete-link clustering and cosine similarity to work the best. Algorithm 3 shows our clustering algorithm. Suppose $sim(\vec{l_i}, \vec{l_j})$ is the cosine similarity between the absorption distributions of refinements r_i and r_j. Then the complete-link similarity between two sets of refinements R_l and R_m is the minimum similarity between two of their respective refinements, i.e.,

$$\mathit{completelink}(R_l, R_m) = \min_{r_i \in R_l, r_j \in R_m} sim(\vec{l_i}, \vec{l_j})$$

The algorithm works by picking the pair of current clusters that have the highest value for complete-link similarity and merges them. This proceeds until only the required number of clusters remain or no more similar clusters are left.

Cosine similarity is effective for comparing two discrete probability distributions. Meanwhile, complete-link clustering avoids chaining and provides guarantees on similarity within each cluster (although it is prone to outliers) [19].

Clustering Dimensionality: The high dimensionality of the absorption distribution vectors could be a concern for our algorithm. Hence, we limited the size of the vectors as follows. First, we only considered up to 80 query refinements. Second, we limited the number of document states off of each refinement to 15. Together, these conditions lead to an upper bound of 1200 on the dimensionality of absorption distribution vectors. These limitations are justified in practice because refinements beyond the top 80 usually have probability mass of less than 0.2%, and document clicks beyond the top 15 are extremely rare and most of the time statistically insignificant. Hence, such filtering not only simplifies clustering, but also eliminates potential "noise".

4. EXPERIMENTS

In this section we evaluate the quality of the clusters produced with our Markov-model based approach (henceforth `MM`). We compare the clusters produced by `MM` against those produced by competing approaches that only use document clicks or only use sessions co-occurrences. The document-click approach (`DC`) clusters refinements that have similar document click sets (Definition 3), while session-query approach (`SQ`) clusters refinements that have similar session co-occurrence queries (Definition 2). Not only do `DC` and `SQ` use the same building blocks as our method, but also represent the two most common approaches to clustering queries (document-based [3, 5, 6, 10, 12, 20, 26] and session-based [7, 8, 9, 11, 13, 15, 25, 27]).

We begin by looking at cluster statistics, and then present the results of a user study of their quality. We then study the trade-off between producing large clusters and more cohesive clusters. Lastly, we consider if the clusters can be used to track user intent and present query suggestions.

4.1 Experimental Settings

Refinement and Document Sets: We used six months of (completely anonymized) search query logs of Google.com to obtain document-click and session co-occurrence data. We consider sessions of 10 minutes duration. For each query q, r_i was considered a refinement, if it followed q in at least 0.2% of the sessions where q was found. We only included the top 80 most frequent refinements of q in the refinement set $R(q)$. Likewise, we only included the 15 most-frequent click documents in the document set $D(q)$.

Implementation: We used the complete-link clustering algorithm with the cosine vector similarity measure for all three approaches. In `DC`, each refinement r_i was represented by a document click vector, with entries for each document d and value proportional to the number of times d was clicked in response to query r_i. In `SQ`, each r_i was represented by a session co-occurrence vector, with entries for each refinement r_j and value proportional to the number of sessions in which r_i and r_j co-occurred. In `MM`, ϵ was set to 0.6, based on the results for a small test set. However, the results only differ marginally for different values of ϵ.

Query Set: We considered the top $10,000$ most popular refined English queries in the search query log. We filtered the ones that had fewer than 80 refinements, leaving us with ≈ 7900 queries, which were used in the rest of this section.

Number of Target Clusters: We clustered the top 80 refinements into 20 clusters. Although there may be fewer

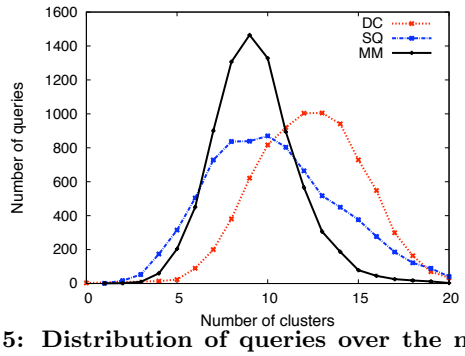

Figure 5: Distribution of queries over the number of clusters that contain their top 20 related queries

than 20 user intents for each query, the document click and session co-occurrence data is sparse for many queries and cannot be used to reliably cluster less popular refinements. As we see in Section 4.4, with fewer than 20 clusters we jeopardize the cohesiveness of clusters.

Note that the complete-link clustering algorithm can alternately have a stopping condition based on a similarity threshold (rather than number of clusters). However, when comparing multiple approaches, we would have to separately pick the best possible similarity threshold for each one, making the comparison tricky. Using a fixed number of clusters, we are asking a fairer comparison of cluster quality given comparable stopping conditions and hence comparable cluster sizes. In Section 4.4, we analyze the results for different numbers of target clusters.

4.2 Cluster Statistics

Before we analyze the quality of the clusters produced by the three methods, we first look at their ability to find similarities between the top refinements of a query. Specifically, given the top 20 refinements of a query, we are interested in the number of clusters each method groups them into. The clustering results are not useful if a method groups the top refinements into either a very small number of clusters or into a very large number of clusters.

Figure 5 shows the distribution of the number of clusters that each method places the top 20 refinements in. The x-axis shows the number of clusters, while the y-axis shows the number of user queries with that number of clusters. Note that the clusters themselves were computed using the top 80 refinements, and hence while there might be 20 clusters overall, only 5 might include top 20 refinements.

As can be seen, MM, in general, clusters the top refinements into fewer clusters. On average, the top 20 refinements are grouped into 9 clusters using MM, 10 clusters using SQ, and 12 clusters using DC. More importantly, there are many more queries under the SQ and DC curves on the right side of the graph than under the MM curve. This indicates that there are many queries for which clustering using MM is feasible, when clustering using SQ or DC may not be possible at all. This result is interesting because it indicates that MM is not merely performing an *average* of SQ and DC by combining the information they rely on.

4.3 User Study

We performed a user study with human evaluators to compare the quality of the clusters generated by MM, DC, and SQ. We randomly selected 400 queries from our original query set and clustered the top 80 refinements for each query using each of MM, DC, and SQ. For each method we created a *cluster group* that lists the clusters for the top 20 refinements of the query, i.e., it only lists the top 20 refinements and the clusters to which they belong. For example, Figure 6 shows the cluster groups generated by MM for some of the queries in the user study. For each query, the cluster groups for the three algorithms are presented side by side to the evaluators.

The evaluators were given three tasks for each query. The first task was to rate each cluster group on the scale: -2 (Poor), -1 (Bad), 0 (Acceptable), 1 (Good), and 2 (Excellent). The second task was to (optionally) indicate if each of the cluster groups was under-clustered or over-clustered. The third task was to indicate which of the three cluster groups were the best (and worst) overall. The task description informed them that determining the quality of a cluster group was highly subjective, and that their decision should depend on how related they believed the refinements in each cluster were and whether they believed that some related refinements had not been clustered together. They were informed that a good cluster group would typically cluster together refinements that referred to the same entity, cluster together different instances of a single underlying concept, and cluster together different related properties of the same entity. Any cluster group that clustered together seemingly unrelated refinements would be a bad one.

The evaluations were conducted on Google's search evaluation framework [22]. Each query was evaluated by three different people and a particular person could evaluate at most 36 different queries. Overall, $400 \times 3 = 1200$ evaluations were performed by about 36 evaluators. For each query, and for each evaluator, the order in which the three cluster groups were presented were selected at random. The evaluators work remotely and are experienced side-by-side testers for Web search results.

Cluster Group Ratings: The *Overall Ratings* columns in Table 1 summarize the results from the first task where each cluster group was rated on a scale of -2 to +2. The *Mean* column presents the rating for each method averaged over all 1200 ratings. MM has a higher mean rating than DC, which has a higher mean rating than SQ. The difference between their ratings is statistically significant: the confidence intervals that cover 95% of the probability mass, assuming Gaussian distribution, for each of the methods do not overlap (the 95% *Bounds* column).

Under/Over Clustering: The *Under Clustered* and *Over Clustered* columns in Table 1 indicate the fraction of queries for which the cluster groups generated by each method were rated to have too many or too few clusters respectively. The *Ratings* columns list the raw number of evaluations (out of 1200), while the *Queries* columns only consider queries (out of 400) in which at least two out of the three evaluators agreed on the rating. The table shows that the results of MM are found to be under-clustered in the least number of cases. DC and SQ are found to be under-clustered in about a third of the cases. MM over-clusters in a few more queries than DC, but much less than SQ. Overall we find that MM was found to under- or over-cluster for only 25% of the queries, much less than that for DC (44%) and SQ (47%).

Best and Worst: The *Best* and *Worst* columns in Table 1 aggregate the results for the best/worst cluster group ratings. As before, the *Ratings* columns have the the raw number of evaluations, while the *Queries* columns show the num-

Table 1: MM is on average rated higher, deemed under- or over-clustered for fewer queries (only 25% combined), and found to be the best for many more queries than DC and SQ (over 400 queries each with 3 evaluators).

Method	Overall Ratings		Under Clustered		Over Clustered		Best		Worst	
	Mean	95% Bounds	Ratings	Queries	Ratings	Queries	Ratings	Queries	Ratings	Queries
MM	0.650	[0.600, 0.700]	282	65 (16%)	211	36 (9%)	616	214 (54%)	166	25 (6%)
DC	0.473	[0.419, 0.527]	479	142 (36%)	203	31 (8%)	356	90 (22%)	329	88 (22%)
SQ	0.056	[-0.003, 0.117]	447	127 (32%)	268	52 (13%)	216	40 (10%)	695	243 (61%)

ber of queries for which at least two of the evaluators agreed on their best (or worst) rated cluster group. Note that for some queries no two evaluators agreed on the best (54 queries) or worst (44 queries) cluster group. Our method, MM, was deemed to be the best cluster group for 54% of the queries, and the worst for only 6% of the queries.

Discussion: We now look at the reasons that are likely to underlie the observed results.

First, we note that document clicks (and hence DC) are very effective in identifying synonymous refinements, e.g., the queries uss enterprise and star trek enterprise that refer to the space ship in the Star Trek series. This is because there are often documents that mention more than one expressions for the same underlying intent. However, they are ineffective in identifying multiple entities of some underlying type, e.g., the queries jupiter, neptune, earth, etc., that refer to planets in the Solar System. This is because there are more documents that are likely to focus on each of the individual entities rather than those that mention all of them. As a result DC is likely to produce more clusters (Figure 5) and hence is deemed to be under-clustering.

Interestingly (and rather surprisingly) DC was also found to be over-clustering for some queries. A closer look indicates that for such queries there is a single document that has almost all of the information about the query, e.g., the Wikipedia page or a home page. This results in multiple refinements being lumped in with the refinement that targets the corresponding single web page, e.g., tiffany necklaces gets lumped with tiffany.com, and michael phelps height is lumped with michael phelps wiki.

Second, session co-occurrences (and hence SQ) are effective in identifying more indirectly related refinements, but less effective in identifying synonyms. For example, for mars, SQ clusters all planets together, but not mars bar and mars candy. However, for some queries SQ clusters refinements that are often posed in close succession in sessions, but they do *not* correspond to real-world notions of similarity, e.g., map of china and population china, or barak obama, hillary clinton, john mccain and mike huckabee (all presidential candidates, but from different political parties). SQ is also sometimes not effective enough to account for the off-topic drift (which is not modeled explicitly), e.g., nasa, mars phoenix, and mars pictures are grouped with the planets clusters.

Third, MM like DC, is able to discover synonymous refinements using document-click information. In addition, the random walk ensures that similarities are spread among the refinements transitively based on the session co-occurrences. Thus, MM is deemed to be under-clustering for very few queries. At the same time, the modeling of off-topic drift and the use of complete-link clustering ensures that it is less likely to over-cluster. Thus, it is able to effectively combine both types of information in a way that the *whole is greater than the sum of parts*. As a result it has both low under- and over-cluster ratings, higher mean ratings, and is deemed the best by a big margin in comparison to DC and SQ.

The examples in Figure 6 illustrate some of the strengths of MM. For england, it is able to separate out the expressions for different entities in the British Isles from other countries in Europe. Likewise, for mama mia, it is able to separate out refinements that refer to the recent movie, the song (along with the performers abba), the broadway musical, and its Las Vegas spin-off. For enterprise, it is able to separate out references to the rental car company from its sales unit and the space ship.

Lastly, we note that the cases in which MM over-clusters appear to be independent from DC and SQ. Specifically, it occurs when one of the top refinements has the same or more possible underlying intents than the original query (e.g., jaguars as a refinement of jaguar). The ambiguous refinement co-occurs with many of the refinements of the original query and during the course of the random walk transitively makes all of them similar, thereby over-clustering. We sequester some obviously ambiguous refinements by marking those that are an edit distance of 1 from the original query. We ignore all incoming edges to these refinements during the random walk and exclude them when invoking `clusterVectors`. After the clustering terminates, we invoke `clusterVectors` again to assign the ambiguous nodes to their most similar cluster. This ensures that they do not play role in transitively making all refinements similar. However, we note that detecting ambiguous refinements is still an area for future work.

4.4 Coverage and Cohesion

We now take a closer look at the tradeoff between cluster sizes, i.e., the number of queries in each cluster, and cluster cohesiveness, i.e., the relatedness of the queries within clusters. We would ideally like to produce larger clusters, but without sacrificing cohesion. We note that coverage and cohesion are analogous to *recall* and *precision* in Information Retrieval systems.

Coverage: To measure the size of a cluster, we assign a weight to each refinement that is proportional to the number of sessions in which it occurs. Specifically, the weight of r_i is $\frac{n(r_i|q)}{\sum_{r_j} n(r_j|q)}$, where $n(r_i|q)$ is the number of sessions in which r_i occurs after q. The coverage of a cluster is the cumulative weight of all the refinements within that cluster. Since we are primarily interested in the most important clusters, we take the cumulative value of coverage for the top k clusters (in decreasing order of coverage) to be a measure of the ability of an algorithm to create large clusters.

Cohesion: To measure the relatedness of the queries within a cluster, we use a method based on *topic similarities*. Briefly, we use a text classifier (called a topic model [23]) to predict the set of topics that each query spans. Since a typical search query only has a few words, in order to get a representative topic distribution vector, we use the snippets of the top-8 search results for the query to be representative of its textual context, and apply the text classifier on each of the snippets. The topic distribution of a query is then an average topic distribution of its snippets and the relatedness of two queries is the *cosine similarity* between their topic distribution vectors. We compute the cohesion for particular

Figure 6: Clusters produced by our MM approach for the top 20 refinements of the queries mars, england, mama mia, and enterprise.

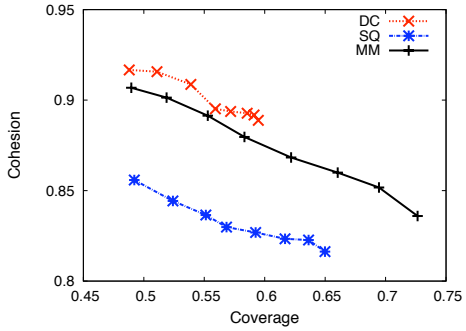

Figure 7: Cohesion-Coverage curves for the top 4 clusters produced by MM, DC, and SQ as we decrease the number of target clusters from 50 to 15.

cluster to be the minimum of the pair-wise cosine similarities between the topic distribution vectors of the different queries in a cluster. We take the average value of cohesion for the top k clusters (again in decreasing order of *coverage*) to be a measure of the cohesiveness of the clusters produced by an algorithm.

Cohension-Coverage Curves: Figure 7 plots the values of cohesion (Y-axis) against coverage (X-axis) for the top 4 clusters as we decrease the number of target clusters from 50 (top left) to 15 (bottom right) in steps of 5. The coverage and cohesion values are averaged over 100 randomly selected queries from our query set. As can be seen, for all three approaches, as we decrease the number of target clusters, the coverage increases, while the cohesion decreases. We find that DC achieves the highest cohesion, but is unable to increase coverage beyond 0.59. This is not surprising. As already highlighted, DC is able to identify synonymous refinements (highest cohesion), but is restricted in its ability to discover other kinds of relations (lower coverage). On the other hand, SQ is not effective in finding synonyms (and hence it starts with lower cohesion), but is able to identify other relations based on session co-occurrences (higher coverage). MM is able to exploit session co-occurrences, transitivity, and document clicks to identify more relations (highest coverage), but is able to account for topic drift and limit transitivity (by complete-link) in order to prevent incorrect decisions (higher cohesion).

Parameter Selection: We can use the cohesion-coverage curve to also select parameters for the clustering algorithm. Based on our target application, we can require a certain level of cohesion among the clusters, and choose the number of target clusters to be the corresponding value. For example, to ensure an average cohesion of 0.85, MM should be set to at least 20 target clusters. Similarly, we could plot cohesion-coverage curve for other parameters, e.g. document escape probability, and use it to select the values.

We note that all the three curves have a slight *knee* when the number of clusters is at 20, i.e., a sharper slope is observed as the target number of clusters decreases to 15. In precision-recall curves this typically indicates a suitable parameter choice that balances their tradeoff. In our case too, we believe that 20 represents a sweet spot in achieving good coverage at acceptable cohesion. Note that this does not mean that all queries have 20 distinct prominent intents. It is rather the result of the facts that there can often be numerous outlier user intents (e.g., the mars manga refers to a Japanese comic strip) and the fact that session co-occurrence and document click information will be sparse for less frequent refinements.

4.5 Intents within a session

Finally, we describe an experiment that measures how users drift between intents. Our purpose is twofold. First, if the clustering algorithm manages to identify intents within a session, then we could use the clusters to suggest better refinements to the user. Second, one of the intuitions underlying our clustering approach is that users do not go back and forth between different intents within a session. Our approach accounts for the possibility that the user's intent may drift (either before satisfying the information need or after), but the assumption is that once the user drifted to a different intent, they typically do not return to the original intent. It is interesting to go back and see whether this behavior is indeed true.

Specifically, we perform the following experiment: Suppose a session has the form $q, r_1, r_2, \ldots, r_n$. Given the results of a clustering algorithm, we denote by $C(r_i)$ the cluster to which the refinement r_i belongs. For each r_i, $2 \leq i \leq n$,

- if $C(r_i) = C(r_{i-1})$, we consider it a *success*, and
- if $C(r_i) \neq C(r_{i-1})$ and there exists a $k < i-1$, such that $C(r_i) = C(r_k)$, then we consider it a *failure*.

In the first case, a refinement suggestion from the present cluster could be deemed as useful. In the second case, the user has drifted to a different intention and back, which means our suggestion would be irrelevant, but also implies that our clustering algorithm may not be identifying intents correctly. The *success rate* is the numbers successes divided by the sum of successes and failures.

We performed the experiment on 500,000 random sessions extracted from the query log. Our results show that MM has the highest success rate: 81.5%. In contrast, SQ and DC has success rates of 75.5% and 51.8%. The fact that DC did

worse than others is not surprising because it does not rely on session data. What is most interesting is that MM beat SQ. SQ is optimized for exploiting co-occurrence of refinements in sessions, but since it does not capture intent as finely as MM, it is likely to be less successful in tracking intents (and hence suggesting queries).

5. RELATED WORK

Our work combines the analysis of document-click and session co-occurrence information. There has been quite a bit of previous work that considered each of these in isolation. In addition, our work is distinguished in that it considers the query refinements in the *context* of an original query.

In addition to the works of Beeferman and Berger [5], Baeza-Yates, et al. [3], and Wen et al. [26], Craswell and Szummer used short backward random walks to find relevant documents [10], while Mei et al. used hitting time of the random walk to pick relevant queries [20]. There have also been several query-expansion methods that rely on the click-through information [6, 12].

In terms of analyzing session data, Cucerzan and Brill ranked query refinements based on the session co-occurrence frequency [11]. Chien and Immorlica mined refinements based on their temporal correlation in sessions [9]. Fonseca et al. used the session co-occurrence information to cluster refinements queries [13], while Boldi et al. used the session co-occurrence data to construct a query-query Markov graph to generate query suggestions [7]. More recently, Wang et al. [24] used session information to cluster global query *qualifiers*, e.g., pictures in mars pictures, and ranked the qualifiers that are most relevant for an input query. In contrast, our approach considers all refinements, not just qualifiers, and clusters them in the context of the original query, not globally. Finally, there have been attempts to specify query suggestions based on the search trail of queries issued by the user [8, 15, 27]. For example, Cao et al. first used document clicks to cluster queries and then separately used session information to match user search trail to a query cluster.

Although we present our clustering objective as a variant of the min k-cut, our graph formulation of the problem can also be similarly mapped to the *correlation clustering* [4]. Correlation clustering, motivated by document clustering, gives an advantage over the classical min k-cut in that it does not require a pre-specified k number of target clusters. In spite of this, the problem is still NP-hard and needs to be approximated [1, 4].

6. CONCLUSION

We describe an algorithm for clustering query refinements that combines information from document-clicks and from user sessions. We formulated the problem as graph-clustering problem on a graph that models user behavior. The graph has a natural interpretation as a Markov model, and we were able to translate it to the clustering problem of the vectors of absorption distributions. Our experiments show that our clustering techniques are clearly favored by users and have the potential of being used for query suggestion.

There are several directions for future work, including developing better treatment of ambiguous queries, and developing methods for leveraging the clusters of refinements to improve the search experience.

Acknowledgements: We thank Abhinandan Das, D. Sivakumar for their insightful thoughts and help with the data.

7. REFERENCES

[1] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: ranking and clustering. In *STOC '05*.
[2] P. Anick. Using terminological feedback for web search refinement: a log-based study. In *SIGIR '03*.
[3] R. Baeza-Yates, C. Hurtado, and M. Mendoza. Query recommendation using query logs in search engines. In *International Workshop on Clustering Information over the Web - EDBT 2004*.
[4] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56(1-3), 2004.
[5] D. Beeferman and A. Berger. Agglomerative clustering of a search engine query log. In *KDD '00*.
[6] B. Billerbeck, F. Scholer, H. E. Williams, and J. Zobel. Query expansion using associated queries. In *CIKM '03*.
[7] P. Boldi, F. Bonchi, C. Castillo, D. Donato, A. Gionis, and S. Vigna. The query-flow graph: model and applications. In *CIKM '08*.
[8] H. Cao, D. Jiang, J. Pei, Q. He, Z. Liao, E. Chen, and H. Li. Context-aware query suggestion by mining click-through and session data. In *KDD '08*.
[9] S. Chien and N. Immorlica. Semantic similarity between search engine queries using temporal correlation. In *WWW '05*.
[10] N. Craswell and M. Szummer. Random walks on the click graph. In *SIGIR '07*.
[11] S. Cucerzan and E. Brill. Extracting semantically related queries by exploiting user session information. Technical report, Microsoft Research, 2005.
[12] H. Cui, J.-R. Wen, J.-Y. Nie, and W.-Y. Ma. Probabilistic query expansion using query logs. In *WWW '02*.
[13] B. M. Fonseca, P. Golgher, B. Pôssas, B. Ribeiro-Neto, and N. Ziviani. Concept-based interactive query expansion. In *CIKM '05*.
[14] O. Goldschmidt and D. S. Hochbaum. Polynomial algorithm for the k-cut problem. In *SFCS '88*.
[15] Q. He, D. Jiang, Z. Liao, S. C. H. Hoi, K. Chang, E.-P. Lim, and H. Li. Web query recommendation via sequential query prediction. In *ICDE '09*.
[16] C.-K. Huang, L.-F. Chien, and Y.-J. Oyang. Relevant term suggestion in interactive web search based on contextual information in query session logs. *J. Am. Soc. Inf. Sci. Technol.*, 54(7), 2003.
[17] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31(3), 1999.
[18] Kosmix. http://kosmix.com.
[19] C. D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
[20] Q. Mei, D. Zhou, and K. Church. Query suggestion using hitting time. In *CIKM '08*.
[21] H. Saran and V. V. Vazirani. Finding k-cuts within twice the optimal. In *SFCS '91*.
[22] Search evaluation at Google. http://googleblog.blogspot.com/2008/09/search-evaluation-at-google.html.
[23] M. Steyvers and T. Griffiths. *Handbook of Latent Semantic Analysis*. Lawrence Erlbaum Associates, 2007.
[24] H. Taylor and S. Karlin. *An introduction to stochastic modeling*. Academic Press, 3rd ed edition, 1994.
[25] X. Wang, D. Chakrabarti, and K. Punera. Mining broad latent query aspects from search sessions. In *KDD '09*.
[26] J.-R. Wen, J.-Y. Nie, and H.-J. Zhang. Clustering user queries of a search engine. In *WWW '01*.
[27] Z. Zhang and O. Nasraoui. Mining search engine query logs for query recommendation. In *WWW '06*.