Task-Aware Query Recommendation

Henry Feild and James Allan
Center for Intelligent Information Retrieval
School of Computer Science
University of Massachusetts
Amherst, MA 01003
{hfeild,allan}@cs.umass.edu

ABSTRACT

When generating query recommendations for a user, a natural approach is to try and leverage not only the user's most recently submitted query, or reference query, but also information about the current search context, such as the user's recent search interactions. We focus on two important classes of queries that make up search contexts: those that address the same information need as the reference query (on-task queries), and those that do not (off-task queries). We analyze the effects on query recommendation performance of using contexts consisting of only on-task queries, only off-task queries, and a mix of the two. Using TREC Session Track data for simulations, we demonstrate that on-task context is helpful on average but can be easily overwhelmed when off-task queries are interleaved—a common situation according to several analyses of commercial search logs. To minimize the impact of off-task queries on recommendation performance, we consider automatic methods of identifying such queries using a state of the art search task identification technique. Our experimental results show that automatic search task identification can eliminate the effect of off-task queries in a mixed context.

We also introduce a novel generalized model for generating recommendations over a search context. While we only consider query text in this study, the model can handle integration over arbitrary user search behavior, such as page visits, dwell times, and query abandonment. In addition, it can be used for other types of recommendation, including personalized web search.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*search process, query formulation*

Keywords

Query recommendation, context-aware recommendation, search task identification

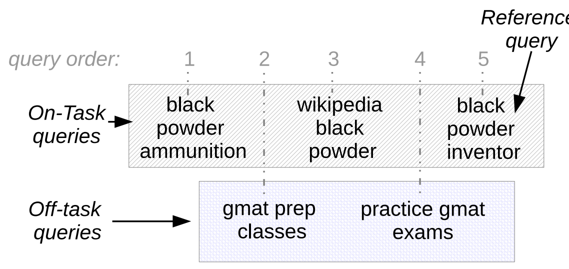

Figure 1: A search context with interleaved tasks.

1. INTRODUCTION

Query recommendation is a common tool used by search engines to assist users in reformulating queries. When an information need, or *task*, requires multiple searches, the sequence of queries form a context around which new queries can be recommended. Figure 1 illustrates a series of queries issued by a user consisting of two tasks: 1) finding information about the history of black powder firearms and 2) preparing for the GMAT standardized test. Given this sequence, our goal is to generate a list of query suggestions with respect to the most recently submitted query, or *reference query*, which is "black powder inventor" in this example. Notice, however, that the user has *interleaved* the two tasks such that no two adjacent queries are part of the same task. If we use the entire context to generate recommendations, two of the queries will be *off-task* with respect to the reference query and three (including the reference query) will be *on-task*. This paper explores the effects that on- and off-task contexts have on query recommendation. While previous work has considered task-aware query recommendation over logged user data, we are not aware of any work that systematically explores the effects of on-task, off-task, and mixed contexts on recommendation performance.

Though the example in Figure 1 may seem an extreme case, consider that Lucchese et al. found 74% of web queries were part of multi-tasking search sessions [15] in a three-month sample of AOL search logs; Jones and Klinkner observed that 17% of tasks were interleaved in a 3-day sample of Yahoo! web searches [10]; and Liao et al. [14] found 30% of sessions contained multiple tasks and 5% of sessions contained interleaved tasks in a sample of half a billion sessions extracted from Bing search logs. In addition, in a labeled sample of 503 AOL user sessions, we found 44% of search tasks consisted of two or more queries (see Figure 2), but there was only a 45% chance that any two adjacent queries

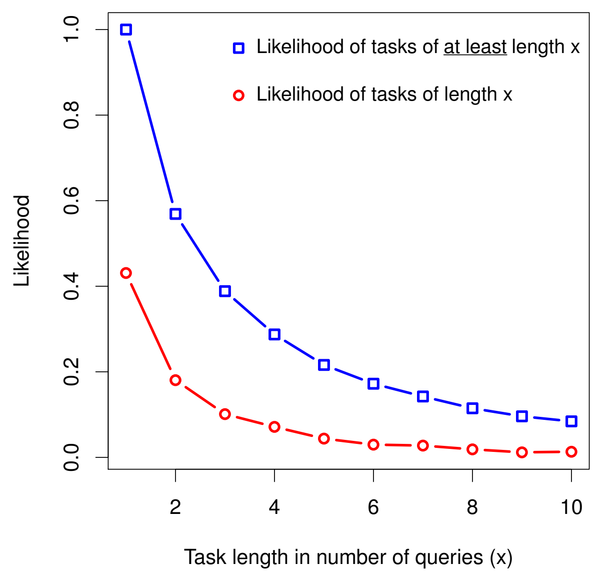

Figure 2: Distribution of tasks lengths observed in a labeled sample of the AOL query log.

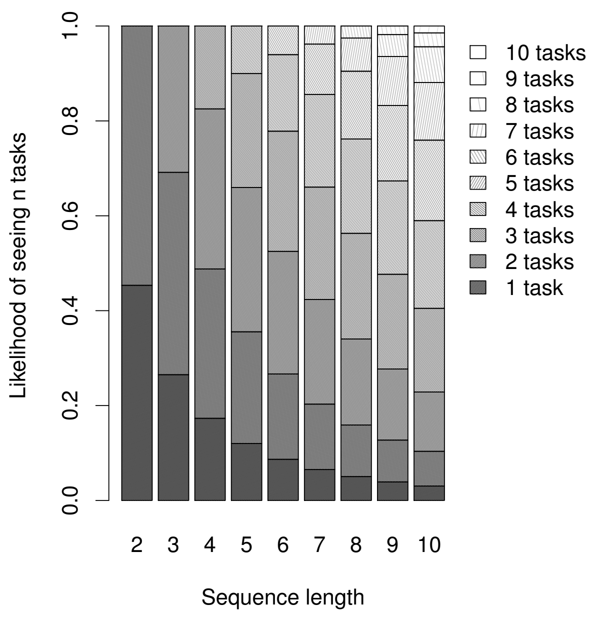

Figure 3: The distribution of seeing n tasks in a sequence of x queries as observed in a labeled sample of the AOL query log.

were part of the same task. Figure 3 shows the likelihood of seeing n tasks in any sequence of x queries, e.g., 10-query sequences typically consist of 3–7 search tasks. This means that a context consisting of the most recent n queries is very likely to consist of sub-contexts for several disjoint tasks, none of which may be a part of the same task as the reference query.

The goal of this paper is to better understand the effects of on-task, off-task, and mixed contexts on query recommendation quality. We also present and analyze several methods for handling mixed contexts. We address four questions concerning query recommendation:

RQ1. How does on-task context affect query recommendation performance?

RQ2. How does off-task context affect query recommendation performance?

RQ3. How does mixed context (on- *and* off-task queries) affect query recommendation performance?

RQ4. How do the following three methods affect query recommendation performance in a mixed context? (a.) using only the reference query, (b.) using the most recent m queries, or (c.) using the most recent m queries with same-task classification scores to weight the influence of each query.

To answer these questions, we perform a number of experiments using simulated search sequences derived from the TREC Session Track. For recommendation, we rely on random walks over a query flow graph formed from a subset of the 2006 AOL query log. We measure query recommendation performance by the quality of the results returned for a recommendation, focusing primarily on mean reciprocal rank (MRR). Our results show that on-task context is usually helpful, while off-task and mixed contexts are extremely harmful. However, automatic search task identification is a reliable way of detecting and discarding off-task queries.

There are four primary contributions of this paper: (1) an analysis of task-aware query recommendation demonstrating the usefulness of on-task query context, (2) an analysis of the impact of automatic search task identification on task-aware recommendation, in which we show the state of the art works very well, (3) an open source data set for evaluating context-aware query recommendation, and (4) a generalized model of combining recommendations across a search context, regardless of the recommendation algorithm.

2. RELATED WORK

Huang, Chien, and Oyang [9] introduced a search log-based query recommendation algorithm that extracts suggestions from search sessions in a query log that appeared similar to the user's current session, thereby incorporating the surrounding search context. They found it outperformed methods that extract suggestions from retrieved documents in many aspects.

Filali et al. [8] presented a probabilistic model for generating rewrites based on an arbitrarily long user search history. Their model interpolates the same-task similarity of a rewrite candidate to the reference query with the average similarity of that candidate to all on-task queries from a user's history, weighted by each query's similarity to the reference query. They found that giving some weight to the history, but most weight to the reference query, did best on a set of manually and automatically labeled data.

Liao et al. [14] explored the effect of task-trails on three applications, including query recommendation. They compare two session-based, two task-based, and two single-query recommendation models and found they retrieve complementary sets of suggestions, though the task-based models provided the higher quality suggestions. To identify tasks, they used an SVM model using features similar to Jones and Klinkner [10], and the weighted connected components clustering method described by Lucchese et al. [15]. Unlike Liao et al., we focus on the effects of on- and off-task context in various mixtures and evaluate on publicly accessible data. Their work is complementary to ours and our findings support theirs concerning the quality of recommendations using on-task recommendations.

Cao et al. [4] introduce a context-aware recommendation system that converts a series of user queries into concept sequences and builds a suffix tree of these from a large query log. To produce recommendations, a concept sequence is looked up in the suffix tree and the common next queries are given as suggestions.

Cao et al. [5] explored an efficient way to train a very large variable length Hidden Markov Model (vlHMM), which considers sequences of queries and clicks in order to produce query and URL recommendations as well as document reranking. The authors trained the vlHMM on a large commercial search log. However, they did not analyze the effects of on- and off-task contexts on recommendation and the vlHMM technique is tied to large search logs, limiting its portability.

Boldi et al. [2] presented a technique for building a query reformulation graph over user sessions, called a query flow graph. They considered two edge weighting techniques: one uses the threshold weight output by a same-task classifier and the other uses the observed likelihood of one query being submitted after the other. They looked at two applications of query flow graphs: (1) identifying sequences of queries that share a common search task and (2) generating query recommendations. The query suggestion component involves random walks and can be configured to consider the most recent n queries.

Several groups have investigated automatic methods for segmenting and clustering user search interactions into search tasks [2, 10, 13, 15, 18]. Most of these depend on machine learning algorithms—logistic regression, support vector machines, and decision trees—trained over large amounts of manually or automatically labeled user data to classify a pair of queries as belonging to the same task or not. We decided to use the Lucchese et al. method [15], which is a heuristic model that depends on lexical and semantic features of the pair of queries being classified. This method is the most recent and was shown to out-perform machine learned models on a sample of AOL data. The other methods are, however, equally applicable to the analysis we conduct.

3. QUERY RECOMMENDATION

We only consider one query recommendation algorithm based largely on the query flow graph (QFG) work of Boldi et al. [2] and the term-query graph (TQGraph) work of Bonchi et al. [3]. We use a query flow graph G in which the vertices V are queries from a query log L and the edges E represent reformulation probabilities. For any vertex $v \in V$, the weights of all outgoing edges must sum to 1. A *reformulation* is defined as an ordered pair of queries (q_i, q_j) such that the pair occurs in a user search session in that order, though not necessarily adjacent. A *session* is defined to be the maximal sequence of queries and result clicks such that no more than t seconds separate any two adjacent events. The outgoing edges of v are normalized across all sessions and users in L.

While we do not require reformulations to be adjacent, Boldi et al. did. By considering all reformulations—adjacent and otherwise—within a session, we expand the coverage of G beyond using adjacent query reformulations only but avoid incorporating as many off-task reformulations as when the scope is a user's entire search history. We assume that reformulations in which q_i and q_j are from different tasks, q_j will be an infrequent follower of q_i, and therefore statistically insignificant among q_i's outgoing edges. In addition, Boldi et al. used a thresholded and normalized chaining probability for the edge weights, but we do not due to the sparseness of our data (the AOL query logs).

To generate recommendations, we rely on a slight adaptation of the query flow graph, called the term-query graph [3]. This adds a layer to the QFG that consists of all terms that occur in queries in the QFG, each of which points to the queries in which it occurs. Given a query q, we find recommendations by first producing random walk scores over all queries in Q for each term $t \in q$.

To compute the random walk with restart for a given term t, we must first create a vector $\mathbf{v}$ of length $|V|$ (i.e., with one element per node in Q). Each element corresponding to a query that contains t is set to 1 and all others are set to 0. This is our *initialization vector*. Next, we must select the probability, c, of restarting our random walk to one of the queries in our initialization vector. Given the adjacency matrix of G, A, and a vector $\mathbf{u}$ that is initially set to $\mathbf{v}$, we then compute the following until convergence:

$$\mathbf{u} = (1 - c)A\mathbf{u} + c\mathbf{v} \tag{1}$$

After convergence, the values in $\mathbf{u}$ are the random walk scores of each corresponding query q' for t. We denote this as the term-level recommendation score $\widehat{\mathbf{r}}_{\text{term}}(q'|t)$.

One issue with using the random walk score for a query is that it favors frequent queries. To address this, Boldi et al. used the geometric mean $\mathbf{r}_{\text{term}}$ of the random walk score $\widehat{\mathbf{r}}_{\text{term}}$ and its normalized score $\overline{\mathbf{r}}_{\text{term}}$. Given an initial query q, the scores for an arbitrary query q' can be computed by:

$$\overline{\mathbf{r}}_{\text{term}}(q'|q) = \frac{\widehat{\mathbf{r}}_{\text{term}}(q'|q)}{\widehat{\mathbf{r}}_{\text{uniform}}(q')} \tag{2}$$

$$\begin{aligned}\mathbf{r}_{\text{term}}(q'|q) &= \sqrt{\widehat{\mathbf{r}}_{\text{term}}(q'|q) \cdot \overline{\mathbf{r}}_{\text{term}}(q'|q)} \\ &= \frac{\widehat{\mathbf{r}}_{\text{term}}(q'|q)}{\sqrt{\widehat{\mathbf{r}}_{\text{uniform}}(q')}} \end{aligned} \tag{3}$$

where $\widehat{\mathbf{r}}_{\text{uniform}}(q')$ is the random walk score produced for q' when the initialization vector $\mathbf{v}$ is uniform.

The final query recommendation vector is computed using a component-wise product of the random walk vectors for each term in the query. Specifically, for each query $q' \in V$, we compute the query-level recommendation score $\mathbf{r}_{\text{query}}(q'|q)$ as follows:

$$\mathbf{r}_{\text{query}}(q'|q) = \prod_{t \in q} \mathbf{r}_{\text{term}}(q'|t) \tag{4}$$

4. CONTEXT AWARE RECOMMENDATION

In this section, we introduce formal definitions of general and task-based contexts as well as the automatic search task identification algorithm used for the experiments.

4.1 Generalized context model

The recommendation models described above, and recommendation algorithms in general, that generate suggestions with respect to a single query can be easily extended to handle additional contextual information. The basic idea is simple: when generating recommendations for a query, consider the search context consisting of the m most recently submitted queries from the user, weighting the influence of each according to some measure of importance. Many functions can be used to measure the importance of a context query. The two we consider in this paper are how far back in a user's search history a query occurs and whether the query appears to be related to a user's current task. However, others may include whether a context query was quickly abandoned or spell corrected, how many results the user visited, the time of day they were visited, and other behavioral aspects. In this section, we introduce a generalized model that makes it easier for us to talk about the various importance functions we are interested in and can be used with additional functions explored in future work.

Assume that we have a search context C that contains all the information about a user's search behavior related to a sequence of m queries, with the m^{th} query, $C[m]$, being the most recently submitted query. Also assume that we have a set of n importance functions, $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_n$ and corresponding weights $\lambda_1, \lambda_2, \ldots, \lambda_n$ that tell us how much weight to give to each of the importance functions. We will represent corresponding functions and weights as tuples in a set $F = \{\langle\lambda_1, \boldsymbol{\theta}_1\rangle\ \langle\lambda_2, \boldsymbol{\theta}_2\rangle\ \ldots, \langle\lambda_n, \boldsymbol{\theta}_n\rangle\}$. We compute the context-aware recommendation score for a query suggestion q' as follows:

$$\mathbf{r}_{\text{context}}(q'|C, F) = \sum_{i=1}^{m} \mathbf{r}_{\text{query}}(q'|C[i]) \sum_{\langle\lambda, \boldsymbol{\theta}\rangle \in F} \lambda \cdot \boldsymbol{\theta}(i, m, C) \tag{5}$$

Each importance function $\boldsymbol{\theta}$, takes three parameters: i, m, C, where i is the position of the query within context C for which importance is being measured and m is the position of the reference query. In this work, the reference query is always the last query of C, but the model does not make the assumption that this is always the case. The $\mathbf{r}_{\text{context}}$ recommendation scoring function scores q' with respect to each query in the context ($C[i]$) and adds that score to the overall score with some weight that is computed as the linear combination of the importance function values for that query.

4.2 Decaying context model

One of the importance functions we consider in this paper is a decaying function, where queries earlier in a user's context are considered less important than more recent queries. As such, queries submitted more recently have a greater influence on recommendation scores. This has the intuitive interpretation that users are less interested in older queries, otherwise they would not have moved on to new queries.

Boldi et al. [2] discussed a decay weighting method for entries in the random walk initialization vector ($\mathbf{v}$ in Eq. 1). They proposed that each query in a search context receive a weight proportional to β^d, where d is the distance in query count from the current query. For example, the second most recent query would get a weight of β^1, because it's one query away from the most recent query.

While the Boldi et al. method is specific to recommendations using random walks, we can transfer their exponential decay function to our model as follows:

$$\mathbf{decay}(i, j, C) = \beta^{j-i} \tag{6}$$

$$\mathbf{r}_{\text{decay}}(q'|C) = \mathbf{r}_{\text{context}}(q'|C, \{\langle 1.0, \mathbf{decay}\rangle\}) \tag{7}$$

4.3 Task context model

While decaying the influence of queries earlier in a search context is a natural importance function, we are also interested in functions that incorporate the degree to which a query is on the same task as the reference query. It is reasonable to assume (an assumption we test) that queries from a search context that are part of the same task should be more helpful in the recommendation process than queries that are not.

As we have stated earlier, Lucchese et al. observed that 74% of web queries are part of multi-task search sessions [15] while Jones and Klinkner found that 17% of tasks are interleaved in web search [10]. Using a labeled sample of the AOL query log, we observed an exponential decrease in the likelihood that the previous m queries are part of the same task as m increases (see Figure 3). This suggests that using the m most recent queries as the the search context for generating recommendations will likely introduce off-topic information, causing recommendations that seem out of place. Therefore, it may be beneficial to identify which queries from that context share the same task as the reference query.

Formally, given a search context C with m queries, we define a task context T to be the maximal subset of queries in C that share a common task to $C[m]$:

$$T \subseteq C \mid \forall\ i \in [1, m], C[i] \in T \iff \mathbf{sametask}(i, m, C) > \tau \tag{8}$$

where $\mathbf{sametask}(i, m, C)$ is a function that outputs a prediction in the range [0,1] as to how likely $C[i]$ and $C[m]$ are to be part of the same search task given C and τ is the decision threshold.

Once we have T, the natural question to pose is how do we use it? One method would be to treat T just as C and use it with the $\mathbf{r}_{\text{decay}}$ function, i.e., $\mathbf{r}_{\text{decay}}(q'|T)$. However, it may be that the off-task context is still useful, just not as useful as T. To support both of these points of view, we can use the following *hard task* recommendation scoring functions:

$$\mathbf{taskdecay}(i, j, C) = \beta^{\mathbf{taskdist}(i,j,C)} \tag{9}$$

$$\mathbf{hardtask}(i, j, C) = \begin{cases} \mathbf{taskdecay} & \text{if } \mathbf{sametask} > \tau, \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

$$\begin{aligned} \mathbf{r}_{\text{hardtask}}(q'|C) = \mathbf{r}_{\text{context}}(q'|C, \{&\langle\lambda, \mathbf{hardtask}\rangle, \\ &\langle 1-\lambda, \mathbf{decay}\rangle\}) \end{aligned} \tag{11}$$

where λ can be used to give more or less weight to the task context and **taskdist** is the number of on-task queries between $C[i]$ and $C[j]$. If we set $\lambda = 1$, we only use the task context, whereas with $\lambda = 0$, we ignore the task context altogether. If we use $\lambda = 0.5$, we use some of the task information, but still allow the greater context to have a presence. Note that we have left off the parameters to **decay** and **sametask** in Eq. 16 for readability.

This approach may work well if one is comfortable with setting a hard threshold τ on the output of the **sametask** function. If, however, we want to provide a mechanism by which we use the output of **sametask** as a confidence, we can use the following *soft task* recommendation scoring functions:

$$\textbf{softtask}(i, j, C) = \textbf{sametask} \cdot \textbf{decay} \tag{12}$$

$$\mathbf{r}_{\text{softtask}}(q'|C) = \mathbf{r}_{\text{context}}(q'|C, \{\langle\lambda, \textbf{softtask}\rangle, \langle 1-\lambda, \textbf{decay}\rangle\}) \tag{13}$$

Here, λ smooths between using and not using the same-task scores to dampen the decay weights. As before, we have left off the parameters to **sametask** and **decay** in Eq. 12.

Two additional models we consider are both variations of what we call *firm task* recommendation, as they combine aspects of both the hard and soft task models. The first, called **firmtask1**, behaves similarly to the soft task model, except that the weight given to any queries with a same task score at or below the threshold τ are weighted as 0. The second, called **firmtask2**, is identical to the hard task model, except that the task classification score is used in addition to the **taskdecay** weight. Mathematically, the firm task recommendation models are described by:

$$\textbf{firmtask1}(i, j, C) = \begin{cases} \textbf{sametask} & \text{if } \textbf{sametask} > \tau, \\ \quad \times \textbf{ decay} & \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

$$\mathbf{r}_{\text{firmtask1}}(q'|C) = \mathbf{r}_{\text{context}}(q'|C, \{\langle\lambda, \textbf{firmtask1}\rangle, \langle 1-\lambda, \textbf{decay}\rangle\}) \tag{15}$$

$$\textbf{firmtask2}(i, j, C) = \begin{cases} \textbf{sametask} & \text{if } \textbf{sametask} > \tau, \\ \quad \times \textbf{ taskdecay} & \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

$$\mathbf{r}_{\text{firmtask2}}(q'|C) = \mathbf{r}_{\text{context}}(q'|C, \{\langle\lambda, \textbf{firmtask2}\rangle, \langle 1-\lambda, \textbf{decay}\rangle\}) \tag{17}$$

Note that unlike the hard task model, the decay component of the **firmtask1** model is affected by every query, not just those above the threshold.

For example, suppose we have a context C with five queries, $q_1, \ldots, q_5$. Relative to the reference query, q_5, suppose applying **sametask** to each query produces the same-task scores [0.4, 0.2, 0.1, 0.95, 1.0]. If we set $\tau = 0.2$, then $T = [q_1, q_4, q_5]$. Using $\beta = 0.8$, notice in Figure 4 how the importance weight of each query in the context changes between using only the decay function (a.) and setting $\lambda = 1$ for the task-aware recommendations (b.–e.). Note that when $\lambda = 0$, the hard, firm, and soft task recommendation scores are equivalent (they all reduce to using the decay-only scoring function).

There are two primary differences between using hard- and soft task recommendation. First, hard task recommendation does not penalize on-task queries that occur prior to a sequence of off-task queries, e.g. in Figure 4, we see that q_1 is on-task and **hardtask** treats it as the first query in a sequence of three: $\beta^{n-1} = 0.8^2$. Conversely, the soft task recommendation model treats q_1 as the first in a sequence of five: $\beta^{m-1} = 0.8^4$.

Second, soft task recommendation can only down-weight a query's importance weight, unlike-hard task recommendation, which we saw can significantly increase the weight of an on-task query further back in the context. At the same time, however, soft task recommendation only allows a query to be given a zero weight if its same-task score is zero. The two firm task models balance these aspects in different ways.

	sametask	=	[.8, .2, .1, .9, 1.0]
a.	$\mathbf{r}_{\text{decay}}$	$\approx$	[.4, .5, .6, .8, 1.0]
b.	$\mathbf{r}_{\text{softtask}}$	$\approx$	[.3, .1, .1, .7, 1.0]
c.	$\mathbf{r}_{\text{firmtask1}}$	=	[.3, .0, .0, .7, 1.0]
d.	$\mathbf{r}_{\text{firmtask2}}$	=	[.5, .0, .0, .7, 1.0]
e.	$\mathbf{r}_{\text{hardtask}}$	$\approx$	[.6, .0, .0, .8, 1.0]

Figure 4: An example of the degree to which each query in a context contributes (right column) given its predicted same-task score (top row) for: (a.) decay only, (b.) soft task, (c.) firm task-1, (d.) firm task-2, and (e) hard task recommendation. We set $\beta = 0.8$ for all, and $\lambda = 1, \tau = 0.2$ for b.–e.

4.4 Automatic search task identification

In Section 3, we discussed how to use information about tasks for query recommendation, but we did not say how to generate the scores. We use the search task identification heuristic described by Luccehse et al. [15]. In deciding if two queries q_i and q_j are part of the same task, we calculate two similarity measures: a lexical and a semantic score, defined as follows.

The lexical scoring function $\mathbf{s}_{\text{lexical}}$ is the average of the Jaccard coefficient between term trigrams extracted from the two queries and one minus the Levenshtein edit distance of the two queries. The score is in the range [0,1]. Two queries that are lexically very similar—ones where a single term has been added, removed, or reordered, or queries that have been spell corrected—should have an $\mathbf{s}_{\text{lexical}}$ score close to one.

The semantic scoring function $\mathbf{s}_{\text{semantic}}$ is made of up two components. The first, $\mathbf{s}_{\text{wikipedia}}(q_i, q_j)$, creates the vectors v_i and v_j consisting of the tf·idf scores of every Wikipedia document relative to q_i and q_j, respectively. The function then returns the cosine similarity between these two vectors. The second component, $\mathbf{s}_{\text{wiktionary}}(q_i, q_j)$ is similarly computed, but over Wiktionary entries. We then set $\mathbf{s}_{\text{semantic}}(q_i, q_j) = \max\big(\mathbf{s}_{\text{wikipedia}}(q_i, q_j), \mathbf{s}_{\text{wiktionary}}(q_i, q_j)\big)$. As with the lexical score, the range of the semantic score is [0,1].

The combined similarity score, $\mathbf{s}$, is defined as follows:

$$\mathbf{s}(q_i, q_j) = \alpha \cdot \mathbf{s}_{\text{lexical}}(q_i, q_j) + (1-\alpha) \cdot \mathbf{s}_{\text{semantic}}(q_i, q_j)$$

We can define a same-task scoring function to use s directly, as follows:

$$\textbf{sametask}_1(i, j, C) = \mathbf{s}(C[i], C[j]) \tag{18}$$

Alternatively, we can run one extra step: single-link clustering over the context C using s as the similarity measure. Clustering allows us to boost the similarity score between two queries that are only indirectly related. Similar to Liao et al. [14], our choice of clustering follows the results of Lucchese et al. [15], who describe a weighted connected components algorithm that is equivalent to single-link clustering with a cutoff of η. After clustering, we use the notation $K_C[q]$ to represent the cluster or task associated with query q in context C; if two queries $q, q' \in C$ are part of the same task, then $K_C[q] = K_C[q']$, otherwise $K_C[q] \neq K_C[q']$. A

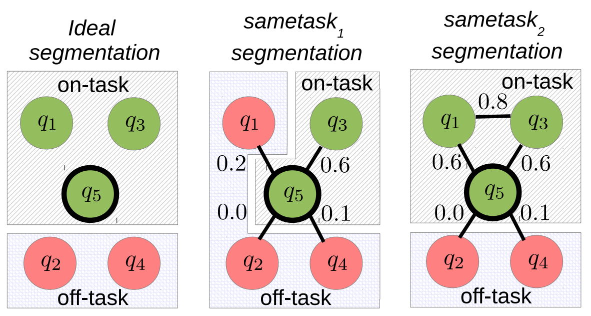

Figure 5: Examples of on-task/off-task segmentations using sametask$_1$ and sametask$_2$ scoring. The reference query, q_5, sits in the bolded center node. Note that the edge (q_1, q_5) goes from 0.2 using sametask$_1$ to 0.6 under sametask$_2$ due to q_1's strong similarity to q_3, which has a similarity score of 0.6 with q_5.

scoring function that uses task clustering is the following:

$$\textbf{sametask}_2(i, j, C) = \max_{\substack{k \in [1,|C|] : \\ k \neq i \wedge \\ K_C[C[k]] = K_C[C[j]]}} \mathbf{s}(C[i], C[k]) \quad (19)$$

Note that **sametask**$_2$ will return the highest similarity between $C[i]$ and any member of $C[j]$'s tasks, excluding $C[i]$. Figure 5 illustrates a case in which **sametask**$_2$ improves over **sametask**$_1$; note, however, that **sametask**$_2$ can also be harmful when an off-task query is found to be similar to an on-task query.

5. EXPERIMENTAL SETUP

In this section, we describe the data, settings, and methodology used for the experiments.

5.1 Constructing a query flow graph

We extracted query reformulations from the 2006 AOL query log, which includes more than 10 million unique queries making up 21 million query instances submitted by 657,426 users between March–April 2006. Considering all ordered pairs from a 30-query sliding window across sessions with a maximum timeout of 26 minutes, we extracted 33,218,915 distinct query reformulations to construct a query flow graph (compared to 18,271,486 if we used only adjacent pairs), ignoring all dash ("-") queries, which correspond to queries that AOL scrubbed or randomly replaced. The inlink and outlink counts of the nodes in the graph both have a median of 2 and a mean of about 5. If we were to use only adjacent reformulations from the logs, the median would be 1 and the mean just under 2.

5.2 Task data

We used the 2010 and 2011 TREC Session Track [11, 12] data to generate task contexts. The 2010 track data contains 136 judged sessions, each with two queries (totaling 272 queries), covering three reformulation types: drift, specialization, and generalization relative to the first search. We ignore the reformulation type. The 2011 track data consists of 76 variable length sessions, 280 queries (average of 3.7 queries per session), and 62 judged topics. Several topics have multiple corresponding sessions. In total, we use all 212 judged sessions from both years. The relevance judgments in both cases are over the ClueWeb09 collection. Our goal is to provide recommendations to retrieve documents relevant to the last query in each session, thus we mark the last query as the reference query.

Each session constitutes a single task, and henceforth we refer to the sessions as tasks. Since the TREC data consists of single tasks, we need some way of simulating the case that multiple tasks are interleaved. We describe our approach for this next.

5.3 Experiments

To answer our four research questions, we use the following set of experiments. Throughout all of these experiments, the baseline is to use only the reference query for generating query recommendations.

Experiment 1. For RQ1, which seeks to understand the effect of including relevant context on recommendation performance, we use each task T from the TREC Session Track and recommend suggestions using the most recent m queries for $m = [1, |T|]$. If incorporating context is helpful, then we should see an improvement as m increases. Note that $m = 1$ is the case in which only the reference query is used.

Experiment 2. To address RQ2, which asks how off-task context affects recommendation performance, we modify the experiment described above to consider a context of $m = [1, |T|]$ queries such that queries 2–$|T|$ are off-task. To capture the randomness of off-task queries, we evaluate over R random samples of off-task contexts (each query is independently sampled from other tasks, excluding those with the same TREC Session Track topic) for each task T and each value of $m > 1$. If off-task context is harmful, we should see a worsening trend in performance as m increases.

Experiment 3. To address RQ3, which asks how query recommendation performance is affected by a context that is a mix of on- and off-task queries, we rely on a simulation of mixed contexts. As we saw in Figure 3, the probability that a sequence of m queries share the same task decreases exponentially as m increases, and so the mixed context assumed in RQ3 is realistic if not typical. We simulate mixed contexts by taking each task T of length n and considering a context window of length $m = [1, n + R]$, where R is the number of off-task queries to add into the context. The last query in the context q_m always corresponds to the last query q_n in T. Queries $q_1, \ldots, q_{m-1}$ consist of a mix of the queries from T and other tasks from the TREC Session Track. The queries from T will always appear in the same order, but not necessarily adjacent.

To incorporate noise, we initially set $C = []$. We select R off-task queries as follows: first, we randomly select an off-topic task, O, from the TREC Session Track and take the first R queries from that task. If $|O| < R$, we randomly selected an addition off-topic task and concatenate its first $R - |O|$ queries to O. We continue the process until $|O| = |R|$. We now randomly interleave T and O, the only rule being that T_n—the reference query—must be the last query in C (an easy rule to adhere to by simply removing T_n before the randomized interleaving, and then concatenating it to the end). For a given value of R, we can perform many randomizations and graph the effect of using the most recent $n + R$ queries to perform query recommendation.

Experiment 4. The final research question, RQ4, asks how mixed contexts should be used in the query recommendation process. We have limited ourselves to consider three possibilities: (a.) using only the reference query (i.e., our baseline throughout these experiments), (b.) using the most recent $n + R$ queries (i.e., the results from Experiment 3), or (c.) incorporating same-task classification scores. Experiment 4 concentrates on (c.) and analyzes the effect of incorporating same-task classification scores during the search context integration process. This is where we will compare the task-aware recommendation models described in the previous section.

5.4 Technical details

For the query recommendation using the TQGraph, we used a restart probability of $c = 0.1$, as was found to be optimal by Bonchi et al. [3]. Note that they refer to the restart value α, where $c = 1 - \alpha$. To increase the speed of our recommendation, we only stored the 100,000 top scoring random walk results for each term. Bonchi et al. [3] found this to have no or very limited effects on performance when used with $c = 0.1$.

For task classification, we used the parameters found optimal by Lucchese et al. [15]: $\eta = 0.2$ (used during task clustering) and $\alpha = 0.5$ (used to weight the semantic and lexical features). We also set $\tau = \eta$ since τ is used in much the same way in the task-aware recommendation models.

To evaluate recommendations, we retrieved documents from ClueWeb09 using the default query likelihood model implemented in Indri 5.3 [17].[1] We removed spam by using the Fusion spam score dataset [6] at a 75th percentile, meaning we only kept the least spammy 25% of documents.[2]

6. RESULTS

In this section, we cover the results of each of the experiments described in Section 5.3. We then discuss the meaning of our findings as well as their broader implications.

6.1 Experimental results

In all experiments, we measured recommendation performance using the mean reciprocal rank (MRR) of ClueWeb09 document retrieved for the top scored recommendation averaged over the 212 TREC Session Track tasks. We found similar trends using normalized discounted cumulative gain (nDCG) and precision at 3, 5, and 10. There are several ways one can calculate relevance over the document sets retrieved for recommendations, such as count any document retrieved in the top ten for any of the context queries as non-relevant (rather harsh), indifferently (resulting in duplicate documents), or by removing all such documents from the result lists of recommendations. We elected to go with the last as it is a reasonable behavior to expect from a context-aware system. We removed documents retrieved for any query in the context, not just those that are on-task. This is a very conservative evaluation and is reflected in the performance metrics.

Experiment 1. With this experiment, our aim was to quantify the effect of on-task query context on recommendation quality. Focusing on the top line with circles in Figure 6, the MRR of the top scored recommendation averaged over the 212 tasks performs better than using only the reference query (middle line with squares). To generate these scores, we used the $\mathbf{r}_{\text{decay}}$ model with $\beta = 0.8$, as set by Boldi et al. [2] in their decay function. For each value of m, if a particular task T has fewer than m queries, the value at $|T|$ is used. The MRR scores are low because for a large number of tasks, none of the methods provide any useful recommendations. We performed evaluations where such tasks were ignored and found that the MRR does indeed increase and the relationship between the methods plotted stays the same. However, in order to ensure comparability with future work, we elected to report on all tasks.

While performance is better on average in Figure 6, the top bar chart in Figure 7 breaks the performance down by the TREC search tasks and we can see that there are many tasks for which on-task context is very helpful, as well as several where it hurts. Note that some of the tasks are not displayed for readability.

Experiment 2. The goal of the second experiment was to ascertain the effect of off-task context on query recommendation. We generated 50 random off-task contexts for

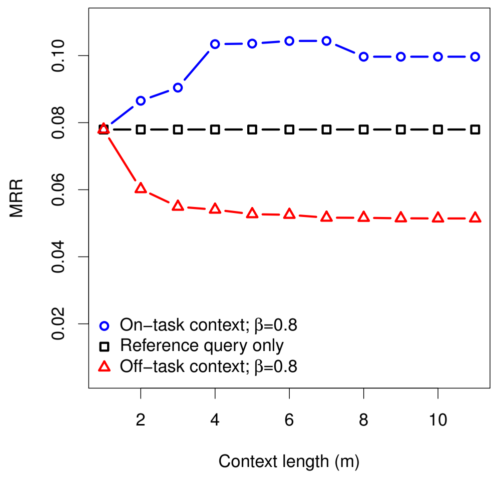

Figure 6: The effect of adding on-task (blue circles) and off-task (red triangles) queries versus only the reference query on recommendation MRR (black squares). MRR is calculated on the top scoring recommendation.

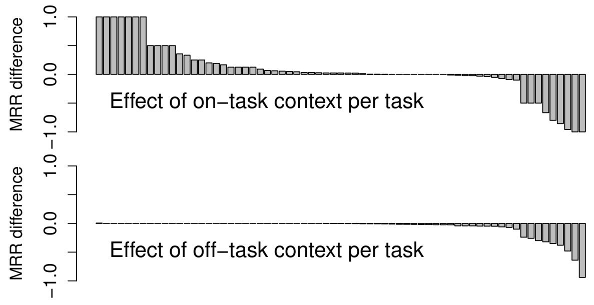

Figure 7: The per session effect of on- and off-task context on the change in MRR of the top scoring recommendation. The y-axis shows the difference between the MRR of using context and using only the reference query. A higher value means context improved MRR. Note that 145 tasks were removed as neither on- nor off-task context had an effect. The bars are not necessarily aligned between the two plots and should not be compared.

[1]http://www.lemurproject.org/indri/
[2]http://plg.uwaterloo.ca/~gvcormac/clueweb09spam/

each task and report the micro-average across all trials. The bottom line with triangles in Figure 6 shows that adding off-task queries under the $\mathbf{r}_{\text{decay}}$ model with $\beta = 0.8$ rapidly decreases recommendation performance for low values of m before more or less leveling off around $m = 5$ (it still decreases, but much slower). Its performance is well below that of the baseline of using only the reference query, making it clear that off task context is extremely detrimental.

Turning to the bottom plot in Figure 7, we see that off-task context has an almost entirely negative effect (there is an ever so slight increase in performance for the task represented by the far left bar). Interestingly, for the severely compromised tasks on the far right, the effect is not as negative as when on-task context hurts. We have not conducted a full analysis to understand this phenomena, but one possible cause is the averaging over 50 trials that takes place for the off-task contexts. We leave investigations into this for future work.

Experiment 3. With Experiment 3, we wanted to understand the effect of mixed contexts—consisting of both on- and off-task queries—on query recommendation performance. As explained earlier, the experiment explores the performance of tasks when R noisy queries are added to the entire set of on-task queries. The bottom line with triangles in Figure 8 shows just this, using $\mathbf{r}_{\text{decay}}$ with $\beta = 0.8$. The far left point, where $R = 0$, lines up with the far right point of the on-task line in Figure 6. We randomly generated 50 noisy contexts per task for each value of R. The solid line shows the micro-averaged MRR over all tasks' samples. The dotted lines on either side show the minimum and maximum values for the micro-average MRR on a set of 1,000 sub-samples (with replacement) of the original 50. As you can see, the bounds indicate relatively low variance of the micro-average across the 212 tasks. There are still certain tasks for which performance is very high or very low (that is, the bounds on the micro-average do not inform us of the variance among tasks).

An important observation from this experiment is that performance dips below that of the baseline when even a *single* off-task query is mixed in. This is quite startling when you consider that the chances of three queries (at $R = 1$, all contexts are of at least length three) in a row belonging to a single task are below 30% (see Figure 3) and that roughly 40% of tasks in the wild are of length three or more (see Figure 2). These results clearly show that blindly incorporating mixed context is a poor method of incorporating context.

Experiment 4. In the final experiment, we hoped to determine the effects of using recommendation models that consider the reference query only, the entire context, or the entire context, but in a task-aware manner. The first two were addressed in the previous experiments, where we learned that using the reference query is more effective than blindly using the entire context. Figure 8 shows the results of using the models we introduced in Section 4.3. We used the same randomizations as in Experiment 3 and likewise generated the minimum and maximum bounds around each model's performance line. For these experiments, $\textbf{sametask}_1$ scores were used to produce same-task scores. We also performed the experiment using $\textbf{sametask}_2$ and found it was comparable. We used $\lambda = 1$ for all task-aware models; setting it to anything less resulted in an extreme degradation of performance.

There are several interesting observations. First, the firm-

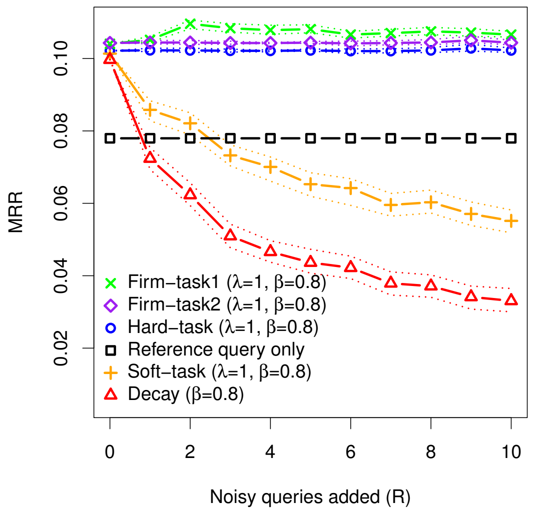

Figure 8: The effect of adding off-task queries to a task context on MRR when same task classification is used and is not used versus only using the reference query (black squares). The $\textbf{sametask}_1$ scoring method is used for all task-aware recommendation models. MRR is calculated on the top scoring recommendation.

task models performed best, though it is likely that the performance of the $\mathbf{r}_{\text{firmtask1}}$ model (the top line with x's) would decrease with larger amounts of noise because the decay function depends on the length of the context, not the number of queries predicted to be on-task. Thus, for on-task queries occurring early on in very large contexts, the decay weight will effectively be 0. You may notice that this model also increases for a bit starting at $R = 2$. This is likely due to the decay function used: since every query in the context, and not just the on-task queries, count toward the distances between an on-task query and the reference query under $\mathbf{r}_{\text{firmtask1}}$, on-task queries are actually down-weighted to a greater degree than in the $\mathbf{r}_{\text{firmtask2}}$ and $\mathbf{r}_{\text{hardtask}}$ models. The graph indicates that this change in weighting is helpful. This also suggests that setting β differently may improve the performance of the other models.

The $\mathbf{r}_{\text{firmtask2}}$ model (diamonds) comes in at a close second and narrowly outperforms $\mathbf{r}_{\text{hardtask}}$ (circles). All three models outperform the baselines—using only the reference query (squares) and $\mathbf{r}_{\text{decay}}$ over the entire context (triangles).

The $\mathbf{r}_{\text{softtask}}$ model, however, performs rather poorly. While it can offer some improvement over using just the reference query for a small amount of noise, once the noise level reaches four off-task queries, it is not longer viable. It does, however, outperform the decay model applied in a task-unaware manner.

Another interesting point is that the performance of the task-aware models is actually better at $R = 0$ than if the known tasks are used. The likely explanation is that the same-task scores prevent on-task queries that are quite different from the reference query from affecting the final recommendation. These kinds of queries may introduce more noise since their only recommendation overlap with the reference query may be generic queries, such as "google". This is not always the case, however. For example, one task consists of the queries ["alan greenspan", "longest serving Federal Reserve Chairman"]. The first query is detected to be

Rank	Reference query	RR	Decay	RR	Hard task ($\lambda = 1$)	RR
1.	google		history of films		black powder sabots	1.00
2.	ebay		the history of european culture of drinking		black powder cannons	0.33
3.	yahoo		the history of european alcohol drinking...		google	
4.	yahoo.com		the history of european alcohol drinking		marlin firearms	
5.	google.com		aircraft crash testing		black powder weapons for sale	0.08

Figure 9: The top 5 suggestions generated from three of the models for the randomly generated context shown in Figure 10. Reciprocal rank (RR) values of 0 are left blank.

No.	Query context	sametask_1
5.	**black powder inventor**	**1.00**
4.	**wikipedia black powder**	**0.52**
3.	us geographic map	0.06
2.	**black powder ammunition**	**0.75**
1.	us political map	0.03

Figure 10: An example of a randomly generated mixed context along with the same-task scores. The top query (No. 5) is the reference query. The bolded queries are on-task.

off-task, however, it is imperative to generate decent recommendations since the reference query generates generic Federal Reserve-related queries and not ones focused on Alan Greenspan.

Overall, though, the same-task classification with a threshold $\tau = 0.2$ worked well. The same-task classification precision relative to the positive class was on average 80%. The precision relative to the negative class varied across each noise level, but was on average 99%. The average accuracy was 93%. The decent same-task classification is why the three models at the top of Figure 8 are so flat.

6.2 Discussion

The results of our experiments demonstrate not only the usefulness of on-task context, but also the extreme impact of off-task and mixed contexts. The results from Experiment 4 suggest that the appropriate model is one that balances a hard threshold to remove any influence from context queries predicted to be off-task, and to weight the importance of the remaining queries by both their distance to the reference query and by the confidence of the same-task classification. Based on the results, we recommend the $\mathbf{r}_{\text{firmtask2}}$ model, since its performance will be consistent regardless of how far back in a user's history we go, unlike $\mathbf{r}_{\text{firmtask1}}$.

We did not see any substantial effects from using task clustering, as Liao et al. [14] used. However, other task identification schemes may perform differently; after all, as we saw in Experiment 4, our task identification method actually caused slight improvements over using the true tasks.

To get a feel for the quality of the recommendations produced generally with the AOL query logs and specifically by different models, consider the randomly generated mixed context in Figure 10. The top five recommendations from three methods for this context are shown in Figure 9. Notice that using only the reference query produces popular queries, none of which are related to the context in the least. Meanwhile, blindly using the context produces suggestions that are swamped by the off-task queries. This is in stark contrast to using the hard task model, which suggests four decent looking suggestions, three of which have non-zeros reciprocal rank values.

Another observation from this example is the amount of noise: very popular queries such as "google" appear in the suggestion lists. This is in part due to not finely tuning the various parameters of the underlying recommendation algorithm we used—something we avoided since the recommendation algorithm was not the focus of this work. It is likely also due to the age and scope of the AOL query log, which is not large compared to the commercial logs commonly used in query recommendation research. Nonetheless, the context-aware recommendation models we presented in Section 4 are compatible with any recommendation algorithm that supplies a sufficiently long, scored list of suggestions. We leave the investigation as to whether performance follows for future work.

Many of the tasks for which task-aware recommendation performed well involved specialization reformulations. Some examples include: *heart rate→slow fast heart rate*, *bobcat tractor→bobcat tractor attachment*, *disneyland hotel→disneyland hotel reviews*, and *elliptical trainer→elliptical trainer benefits*. One possible reason for this is that incorporating recommendations for a context query that is a subset of the reference query focuses the final recommendations on the most important concepts.

None of the experiments used notions of temporal sessions or complete user histories. We did this mainly because the mixed contexts were generated and not from actual user logs where context windows could be tied to temporal boundaries, e.g., two-day windows. We believe that by focusing on factors such as on- and off-task queries, we struck at the core questions in this space. We leave testing whether the results from these experiments port to real user data to future work, but we believe they will, especially given the results of related studies, such as those conducted by Liao et al. [14] and Filali et al. [8].

7. SUMMARY AND FUTURE WORK

In this work, we investigated four key research questions surrounding context-aware query recommendation and the effects of on- and off-task recommendation: (RQ1) How does on-task context affect recommendation quality relative to using only the most recently used query? (RQ2) How does off-task context affect recommendation quality? (RQ3) How does mixed context consisting of on- and off-task queries affect recommendation quality? and (RQ4) How do task-aware recommendation models affect performance given a mixed context?

We designed four sets of experiments that used random walks over a term-query graph of the 2006 AOL query logs for recommendations and the 2010–2011 TREC Session Track data for tasks. We evaluated all models using micro-averaged mean reciprocal rank (MRR) over the 212 TREC tasks and used 50 trials when randomization was needed. The results of our experiments show that on-task context is, on average, very helpful, but can sometimes degrade performance substantially. On the other hand, off-task context only degrades performance. We found that leveraging mixed contexts, when used without regard of the task corresponding to each of its constituent queries, reduced the quality of recommendations. Finally, we demonstrated the effectiveness of four task-aware models, which rely on a state-of-the-art search task identification algorithm. Three of the four models out-performed using only the most recently submitted query when up to ten off-task queries were added. Results suggest that two of those models would maintain their accuracy at large levels of noise since their weighting schemes completely ignore any query classified as off-task.

In many ways these results are not surprising: incorporating relevant queries improves query accuracy just as relevance feedback improves document ranking. What is surprising, though, is the extraordinary sensitivity of these approaches to off-task queries. Even making a single mistake and including an off-task query can shift many approaches in the wrong direction, away from relevant documents. This work takes a methodical look at the relative impact of on- and off-task queries and provides a deeper understanding of their inter-relationships than had been reported previously.

In addition, our findings rely on publicly accessible datasets, which eases others' efforts in validating and reproducing our results. Furthermore, it allows others to compare directly with our results if they so desire. The simulated contexts and corresponding recommendations used in our experiments are available on our website.[3]

There are many avenues of future work. One element to explore is the portability of our findings to real user search behavior over many more tasks. Another is to consider other recommendation algorithms, including ones not dependent on query logs [1] and ones that use other user behavior [7, 19]. We only considered the effectiveness of query recommendations to address the current task, but other measures of suggestion quality exists [16], such as attractiveness and diversity; exploring these other evaluations would provide a fuller picture of the value of task-aware recommendation. Other future directions include expanding the analysis to other applications, such as website or task recommendation [20].

8. ACKNOWLEDGMENTS

This work was supported in part by the Center for Intelligent Information Retrieval and in part by NSF grant #IIS-0910884. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsor.

References

[1] S. Bhatia, D. Majumdar, and P. Mitra. Query suggestions in the absence of query logs. In *Proc. of SIGIR*, pages 795–804, 2011.

[2] P. Boldi, F. Bonchi, C. Castillo, D. Donato, A. Gionis, and S. Vigna. The query-flow graph: model and applications. In *Proc. of CIKM*, pages 609–618, 2008.

[3] F. Bonchi, R. Perego, F. Silvestri, H. Vahabi, and R. Venturini. Efficient query recommendations in the long tail via center-piece subgraphs. In *Proc. of SIGIR*, pages 345–354, 2012.

[4] 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 *Proceeding of KDD*, pages 875–883, 2008.

[5] H. Cao, D. Jiang, J. Pei, E. Chen, and H. Li. Towards context-aware search by learning a very large variable length hidden Markov model from search logs. In *Proc. of WWW*, pages 191–200, 2009.

[6] G. Cormack, M. Smucker, and C. Clarke. Efficient and effective spam filtering and re-ranking for large web datasets. *Information retrieval*, 14(5):441–465, 2011.

[7] S. Cucerzan and R. W. White. Query suggestion based on user landing pages. In *Proc. of SIGIR*, pages 875–876, 2007.

[8] K. Filali, A. Nair, and C. Leggetter. Transitive history-based query disambiguation for query reformulation. In *Proc. of SIGIR*, pages 849–850, 2010.

[9] C. Huang, L. Chien, and Y. Oyang. Relevant term suggestion in interactive web search based on contextual information in query session logs. *JASIST*, 54(7):638–649, 2003.

[10] R. Jones and K. L. Klinkner. Beyond the session timeout: automatic hierarchical segmentation of search topics in query logs. *Proceedings of CIKM 2008*, 2008.

[11] E. Kanoulas, P. Clough, B. Carterette, and M. Sanderson. Session track at TREC 2010. In *Proc. of the SIGIR Workshop on the Simulation of Interaction*, pages 13–14, 2010.

[12] E. Kanoulas, B. Carterette, M. Hall, P. Clough, and M. Sanderson. Overview of the TREC 2011 Session Track. In *Proc. of TREC*, 2011.

[13] A. Kotov, P. N. Bennett, R. W. White, S. T. Dumais, and J. Teevan. Modeling and analysis of cross-session search tasks. In *Proc. of SIGIR*, pages 5–14, 2011.

[14] Z. Liao, Y. Song, L.-w. He, and Y. Huang. Evaluating the effectiveness of search task trails. In *Proc. of WWW*, pages 489–498, 2012.

[15] C. Lucchese, S. Orlando, R. Perego, F. Silvestri, and G. Tolomei. Identifying task-based sessions in search engine query logs. In *Proc. of WSDM*, pages 277–286, 2011.

[16] Z. Ma, Y. Chen, R. Song, T. Sakai, J. Lu, and J. Wen. New assessment criteria for query suggestion. In *Proc. of SIGIR*, pages 1109–1110, 2012.

[17] D. Metzler and W. Croft. Combining the language model and inference network approaches to retrieval. *Information processing & management*, 40(5):735–750, 2004.

[18] F. Radlinski and T. Joachims. Query chains: learning to rank from implicit feedback. In *Proc. of KDD*, pages 239–248, 2005.

[19] Y. Song and L.-w. He. Optimal rare query suggestion with implicit user feedback. In *Proc. of WWW*, pages 901–910, 2010.

[20] G. Tolomei, S. Orlando, and F. Silvestri. Towards a task-based search and recommender systems. In *ICDEW*, pages 333–336, 2010.

[3]`http://ciir.cs.umass.edu/downloads/task-aware-query-recommendation`