Multiple Queries as Bandit Arms

Cheng Li
School of Information
University of Michigan
Ann Arbor, MI, USA
lichengz@umich.edu

Paul Resnick
School of Information
University of Michigan
Ann Arbor, MI, USA
presnick@umich.edu

Qiaozhu Mei
School of Information
University of Michigan
Ann Arbor, MI, USA
qmei@umich.edu

ABSTRACT

Existing retrieval systems rely on a single active query to pull documents from the index. Relevance feedback may be used to iteratively refine the query, but only one query is active at a time. If the user's information need has multiple aspects, the query must represent the union of these aspects. We consider a new paradigm of retrieval where multiple queries are kept "active" simultaneously. In the presence of rate limits, the active queries take turns accessing the index to retrieve another "page" of results. Turns are assigned by a multi-armed bandit based on user feedback. This allows the system to explore which queries return more relevant results and to exploit the best ones. In empirical tests, query pools outperform solo, combined queries. Significant improvement is observed both when the subtopic queries are known in advance and when the queries are generated in a user-interactive process.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Relevance Feedback

Keywords

Query pooling; multi-armed bandits

1. INTRODUCTION

In the age of big data, most users of an information retrieval systems do not own the document indices or have infinite resources to examine all possible results. Instead, they rely on rate limited search services to pull relevant documents from the index. These search services, such as commercial search engines, normally take as input a single, length-limited query and output a certain number of documents at a time. This approach is sufficient when the information need of the user can be clearly described in a single query and when the information need can be satisfied by a few relevant documents ranked near the top.

This common practice becomes less efficient when the user has a more complex information need, which may contain multiple aspects (or subtopics) or may evolve during the search process. For example, to market a new product an analyst needs to gather diverse aspects about the product and its competitors; to write a literature review a researcher often surveys several lines of related work and may discover new relevant topics on the fly; to investigate a rumor a social scientist may explore everything including its origin, its variations, and people's attitudes towards it. In all these scenarios, there exist multiple, diverse aspects of the user's information need. Some of these aspects are known upfront, and others may emerge during the search process. It is difficult to precisely describe all these aspects within a single query - even with the help of an intelligent search engine. Indeed, attempting to combine multiple subtopics into a single query may either make it overly lengthy (thus exceeding the maximum size of a query that a search engine can efficiently handle) or introduce unnecessary ambiguity. For example, combining two subtopics "*language, modeling*" and "*learning, parameters*" into one query may confuse the search system and retrieve irrelevant results about "*language learning*."

Existing work approaches this challenge in multiple ways. Some engage the user into an interactive process and refine the query based on her feedback about the results; some rely on smart post-processing of the retrieved documents, for example through clustering or active learning; and others apply both. In all these approaches, there is only one "active" query at any time - the search engine either requests a "deeper" set of results of the current query or replaces previous queries with a new one. How to construct such an effective single query is still a fundamental challenge.

In this paper, we explore a different solution, a solution which releases the search engine from the burden of constructing such a query. Instead of composing a single, complex query, the main idea is to keep multiple, simpler queries "active" at the **same** time. Without being replaced by one another, these queries take turns to pull documents from the document index. Based on the user's feedback, queries that appear to be more effective get more turns than others. When every query covers a different aspect of the information need, a "pool" of simple queries can be more effective than a single complex query.

We cast exploration and exploitation of a query pool as a multi-armed bandit problem and propose algorithms to assign turns to queries. In scenarios where new queries are generated during the search process, new arms are introduced to the bandit. Empirical experiments show that when equipped with bandits, a query pool is superior to a single combined query subject to the same rate limits. Significant improvement is observed both when the subtopic queries are known in advance and when the queries are generated in a user-interactive process. Retrieval performance can be further improved when query pooling is blended with effective post-processing techniques such as results diversification.

It is important to note that pooling queries as bandit arms is different from two existing settings: session search and search result diversification. In session search [5], although the search engine

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CIKM'16 , October 24-28, 2016, Indianapolis, IN, USA
© 2016 ACM. ISBN 978-1-4503-4073-1/16/10. . . $15.00
DOI: http://dx.doi.org/10.1145/2983323.2983816

learns from a set of queries and user feedback in the same session, only the newest query is "active" and the goal is to maximize the relevance of the results retrieved by the single final query. Search result diversification [24] on the other hand is a post-processing approach, where the goal is to rerank the list of retrieved document (by a query) based on the possible subtopics. Alternatively, query pooling keeps multiple queries active simultaneously, which addresses the limitation of single queries. It does not require knowledge of the actual retrieval algorithm and thus can be launched as a general interface between the user and a black-box search engine.

2. RELATED WORK

The key idea of our study is to pool multiple queries (keep them active at the same time) and assign turns using multi-armed bandits, which is a classical method of reinforcement learning. To the best of our knowledge, this is a new approach to interfacing a user with a search engine. Below we discuss the existing literature related to query pooling and applications of reinforcement learning to information retrieval.

2.1 Query Pooling

Despite the rapid developments of retrieval algorithms, result post-processing techniques, and search interfaces, existing information retrieval systems rely on a single active query to find relevant documents. To deal with a complex information need, relevance feedback techniques have been developed to update and replace the initial query [23]. A relevant scenario is session search, where a user keeps modifying queries to find documents that fulfill her information need [5]. Methods have been proposed to reformulate a user's next query in a session through techniques such as structured queries [7], combined query language models [11], and machine learning from clickthrough and preceding queries [26].

Complex information needs are also addressed through subtopic retrieval, which reranks documents retrieved by the initial query. The Maximal Marginal Relevance (MMR) method [4] selects one candidate document at a time by balancing its similarity to the query and its dissimilarity to already selected documents. Zhai et al. [34] selected documents with high divergence from one language model to another based on risk minimization. Other methods explicitly identify aspects of a query using taxonomies [1], query reformulations [22], and multiple external resources [9].

Both relevance feedback (including session search) and subtopic retrieval (including result diversification) techniques deal with results returned by one single query. In session search, the newest query replaces all preceding queries; in result diversification, the original query is not updated.

An interactive retrieval system proposed in [16] does pool the results retrieved by multiple queries, but again only one query is active at any given time. A related idea can also be found in document pooling, a technique used to build test collections to evaluate retrieval methods [13]. In pooling, documents to be judged are collected by taking the union of the top ranked documents given by a variety of retrieval systems. Bandits have been used to select which retrieval method to collect documents from [18]. Document pooling considers query results, rather than queries, making the task much easier – there is no need to handle query updates or new queries. This work explores a new idea to keep multiple active queries which take turns to retrieve the next page of results.

2.2 Reinforcement Learning in Retrieval

The application of reinforcement learning techniques (including multi-armed bandits) to information retrieval problems has received considerable attention recently. These techniques provide a principled trade-off between exploration and exploitation, and therefore are suitable for problems such as recommendation. For example, contextual bandits have been applied to news recommendation [17], where news articles are viewed as arms or actions and user reactions as rewards. Hsieh et al. [10] proposed an algorithm based on Thompson sampling for the problem of query suggestion, helping shoppers refine their queries on e-commerce sites. In this setting, candidate queries are modeled as arms while user clicks are modeled as rewards. Although they also treat queries as arms, the task is quite different from ours: their bandit only decides the display order of queries, and which query to launch next is completely decided by the user. In addition, their bandit can only deal with a fixed set of queries and cannot handle new ones.

When implicit or explicit user feedback is available, document ranking can be also formulated as a bandit problem. Researchers have studied the use of either documents or document-selection strategies as arms to maximize the overall positive feedback [3, 28].

In session search, users issue multiple queries in a sequence to fulfill a single information need. Since there is a sequence of actions from a user, stochastic models with states (such as Partially Observable Markov Decision Processes (POMDP) [30]) can be employed to solve this problem. Luo et al. [20] proposed to model session search as a dual-agent game between the user and the search engine. Users can take actions to add and remove query terms, while a search engine can change term weights and adjust search algorithms. An investigation of how to design states, actions, and rewards within a POMDP framework is conducted in [19]. A scenario similar to session search is multi-page search, where users request one page after another for the same query. How to present documents on the next page depending on the current user feedback is studied both with Bayesian models [12] and with POMDP [29].

The problem we study differs from session search and multi-page search as we do not require control of the search engine. All we need is a search API with rate limits on queries, which can be called as a black box. With these constraints, our objective is to retrieve as many relevant documents as possible with a limited number of calls to the black box, rather than presenting relevant documents in some order on one page. Correspondingly, our actions would be going one page deeper for one of the existing queries, or trying a new query. As we will show in the experiments, document re-ranking can be used to post-process the documents retrieved by the query pool and further improve retrieval performance.

3. METHODS

Users often do not own the document collection and can only access it through centralized search services, subject to rate limits. Even with an in-house search engine, it still makes sense to limit the number of results retrieved for a query and decide based on this limited set of results whether to continue retrieving more or to request results for a different query. We will refer to each request for another page of results for some query as a search API call.

In the following subsections, we first offer intuitions as to why a query pool is better than a single query in handling complex information needs using a rate-limited search service. Then, we introduce bandits to manage the use of queries in the pool. Last, we present an algorithm to tackle the situation where the information need evolves and new queries are generated during the retrieval process.

3.1 Query Pools

A complex information need usually contains multiple aspects or subtopics. There are scenarios where subtopics of an information need are known in advance. For example, an experienced researcher doing literature review knows the general subtopics related to her target. Another example comes from query recommendation. When a user issues a query, the search engine, based on search logs, can return a set of suggested queries, from which the user could pick one or more relevant ones. If these queries are refined and diverse, they can be viewed as subtopics.

Even if there is no a priori way to obtain the subtopics, results of the first few retrievals could help us discover them. More specifically, one can first use active feedback [27], a relevance feedback method that chooses diverse results for user judgments, to collect labeled documents. When the number of positively labeled documents reaches a threshold, these positive documents will be clustered, with one cluster representing one subtopic. Given these clusters, any relevance feedback techniques could be used to generate a query out of each cluster of documents.

3.1.1 *Query Pools vs. Single Queries*

Broadly speaking, there are two general strategies to request results from a search engine when one has an information need with multiple subtopics. The first strategy is to use a single complex query to cover all subtopics. The second strategy is to construct a pool of simple queries, one for each subtopic. A query pool has considerable advantage over a single query, which we intuitively illustrate in Figure 1. In the figure, each subtopic delineates a (latent) region of relevant documents in an information space. These regions, from different subtopics of the same information need, could either overlap or lie far apart. A query retrieves documents close to it in the information space, thus covering a (grey) area surrounding it. As we request the search engine to return more results for a query, the radius of its covered area increases. Given a certain rate limit (e.g., the total pages of results returned), a single circle (or more generally hyper-sphere) can cover a set of oddly shaped, separated positive areas only by also covering parts of the space that contain non-relevant documents. A set of smaller circles can more efficiently cover the relevant documents (positive regions) while avoiding the irrelevant ones. One tradeoff is that if the circles overlap each other, the same document may be retrieved multiple times. There is also a major challenge in pooling: deciding which queries should take more turns (to grow their covered areas).

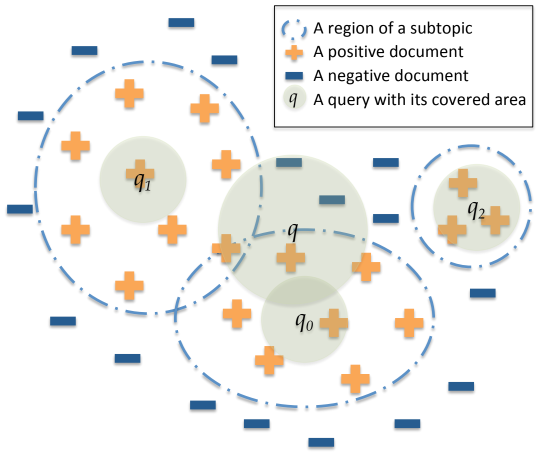


Figure 1: An example of the information space. It is hard to formulate a single query q that perfectly covers the three regions of relevant documents. In contrast, the query set $\{q_0, q_1, q_2\}$ can cover these regions more easily. The sum of areas covered by $\{q_0, q_1, q_2\}$ equals the area of the single query q, as the set of queries have to share the search API calls.

3.2 Bandits for Choosing Queries from a Pool

Even when subtopics are known, it is hard for users to estimate which subtopics will lead to more relevant results. Looking at Figure 1, queries q_0 and q_1 should be explored more often than q_2. Putting too much effort on q_2 would bring in many irrelevant documents. Therefore it is important to optimize the sequence of queries that get to retrieve their next pages of results. Intuitively, the number of relevant documents obtained from past result pages of each query provides a hint on which query to explore next. If we treat relevant documents as rewards, multi-armed bandit algorithms [15] can be a natural fit for this problem, as they try to maximize the expected sum of rewards earned through a sequence of arm pulls. Below we first introduce the classical setting of the bandit problem and then discuss how to formulate our task as a bandit.

3.2.1 *The multi-armed bandit problem*

The bandit problem assumes that at each time t, the player chooses an arm $a_t \in \{1, ..., M\}$ to play according to a policy π based on past plays and rewards, which obtains a reward $X_t(a_t)$. The sequence of rewards for a particular arm are drawn from a sequence of probability distributions that are unknown to the player. For simplicity of notation, assume that the reward distribution for each arm is stationary, independent of the number of times it has been pulled (we will relax this assumption later). Let $\mu(i)$ be the expectation of reward for arm i. The optimal policy π^* always plays the arm i^* with largest expected reward

$$\mu^* = \max_i \mu(i), \; i^* = \arg\max_i \mu(i) \tag{1}$$

A policy π is evaluated by *regret* in T plays, which is defined with respect to the optimal policy π^*. Denote $N_t(i) = \sum_{s=1}^t \mathbb{1}_{a_s=i}$ the number of times arm i has been played in the first t rounds. The expected regret after T plays is:

$$\mathbb{E}_\pi \left[\sum_{t=1}^{T} (\mu^* - \mu(a_t)) \right] = T\mu^* - \mathbb{E}_\pi \sum_{i=1}^{M} N_T(i)\mu(i) \tag{2}$$

One popular policy for picking the next bandit arm is UCB-1 [2], which has a provable bound on the expected regret. At time t, it chooses the arm:

$$a_t = \arg\max_i \left\{ \bar{X}_t(i) + c_t(i) \right\}, \tag{3}$$

where $\bar{X}_t(i) = \frac{\sum_{s=1}^t X_s(i)\mathbb{1}_{a_s=i}}{N_t(i)}$, $c_t(i) = c\sqrt{\frac{\log(t)}{N_t(i)}}$, and c is a constant.

The above policy works for the condition that rewards $\{X_t(i)\}$ for arm i come from the same stationary distribution. For the non-stationary case, sliding-window UCB (SW-UCB) [6] considers only the last τ plays:

$$a_t = \arg\max_i \left\{ \bar{X}_t(\tau, i) + c_t(\tau, i) \right\}, \tag{4}$$

where $\bar{X}_t(\tau, i)$ is

$$\bar{X}_t(\tau, i) = \frac{\sum_{s=t-\tau+1}^{t} X_s(i)\mathbb{1}_{a_s=i}}{N_t(\tau, i)}, \; N_t(\tau, i) = \sum_{s=t-\tau+1}^{t} \mathbb{1}_{a_s=i} \tag{5}$$

and $c_t(\tau, i)$ is given by

$$c_t(\tau, i) = c\sqrt{\frac{\log(\min(t, \tau))}{N_t(\tau, i)}} \tag{6}$$

3.2.2 *Formulating the bandit*

To formulate the search task as a bandit problem, we need to define arms and rewards. Each query in the pool is an arm, and pulling an arm corresponds to retrieving the next page of this query. The definition of reward is somewhat tricky when the same document can be retrieved by multiple queries. If we define the reward as the percentage of documents that are **new** and relevant in a page, a problem of *reward dependence* occurs, where arms played earlier will be strongly favored. This is because early arms in general have a better chance to fetch new documents. If we define the reward as the percentage of relevant documents, regardless of whether they have been retrieved or not, another problem of *arm dependence* emerges. In the extreme case, suppose we have two identical queries; each of them will be pulled half of the time, while the best choice would be to discard one of them. We choose the latter reward function, which risks wasting some time on multiple retrievals but prevents missed opportunities where a highly precise query is overlooked because its first page of results have previously been retrieved. We strive to limit overlap between queries.

Note that with this formulation, the reward of a specific arm is not sampled from a stationary distribution. The number of relevant documents usually goes down when we go to the next page of a query. Since it is more important to trust recent rewards, we use the sliding-window UCB (Equation 4) to select the next query.

3.3 Bandits with New Queries

When a set of queries are known in advance, the exploration and exploitation of them can be handled by a classical bandit algorithm. However, in many cases the subtopics of the information need are not defined upfront but are gradually revealed during the search process. In this subsection, we consider the scenario where new queries are generated through an interactive retrieval process, through techniques such as relevance feedback [23]. Instead of replacing the old queries, a new arm of query is added to the bandit.

3.3.1 *Reward Estimation for New Queries*

As a requirement to apply the UCB policy, classical bandit algorithms assume that there is a fixed number of arms and each arm has to be pulled once for an initial estimate of reward. When new queries are generated frequently (e.g., in an iterative relevance feedback process), if we always pull the new arm to get its initial estimate, we will have fewer turns to play existing arms. In an extreme case where a new query is always generated after receiving the user's feedback, previous queries will never be revisited.

We propose a new algorithm to solve this problem that is inspired by Monte Carlo tree search [31], where the UCB score of a state is estimated by similar states. In our case, the UCB score of a new query is estimated by its similarity to existing queries in the pool $\mathcal{Q}$. Formally, let $R_t(i) = \sum_{t-\tau+1}^{t} X_s(i)\mathbb{1}_{a_s=i}$ be the return, or the sum of rewards to the i-th arm in the last τ rounds. For a new query q_n, we introduce a new arm n and estimate its return, number of plays, and UCB score by:

$$\begin{aligned} \hat{R}_t(n) &= \sum_i (K_{i,n} * R_t(i)) \\ \hat{N}_t(n) &= \frac{\sum_i (K_{i,n} * N_t(\tau, i))}{\sqrt{\sum_i (K_{i,n})}} \qquad (7) \\ U\hat{C}B_t(n) &= \frac{\hat{R}_t(n)}{\hat{N}_t(n)} + \hat{c}\sqrt{\frac{\log(\min(t,\tau) + \hat{N}_t(n))}{\hat{N}_t(n)}}, \end{aligned}$$

where $\hat{c}$ is a constant controlling the tendency to explore the new arm and $K_{i,n}$ is a nonnegative valued kernel function that measures the similarity between arm i and n. Treating a query as a bag of words, we choose the Gaussian kernel $K_{i,n} = \exp(-\frac{\|i,n\|_2}{\sigma})$, where σ is the Gaussian width parameter.

With this new algorithm, the initial reward of a new query can be estimated without actually submitting it to the search engine. Below we introduce a concrete case, an interactive search process in which new queries are continuously added to the pool and new arms are added to the bandit.

3.3.2 *A Case Study: ReQ-ReC with Bandits*

We consider the ReQ-ReC framework proposed in Li et al [16]. ReQ-ReC is an interactive retrieval process that tries to achieve high-recall for a complex information need. It employs a double-loop design that distributes the burden of maximizing recall and precision to a query generator and a classifier. In an outer-loop, the query generator generates a new query (ReQuery) according to the user's feedback obtained so far, adds it into a query set $\mathcal{Q}$, and merges newly retrieved documents into a pool of documents. In an inner-loop, the classifier selects documents from the pool to obtain the user's judgments via active learning [25], accumulates labeled documents, and iteratively improves its classification performance on retrieved documents (ReClassify). The overall process goes on by alternating between the query enhancement loop (to increase recall) and the classifier refinement loop (to increase precision), until stop criteria are met.

The original ReQ-ReC framework does not consider rate limits of queries. All results retrieved by a query are added into the pool at once and therefore there is no need to revisit previous queries. We incorporate bandits into the ReQ-ReC framework to accommodate rate limits of queries, a practical setting when interacting with a real search API. All previous queries are kept alive in the pool and every query only submits one API call (i.e., retrieves one page of results) at a time. The arms and rewards are defined similarly as in Section 3.2, except that now the reward is estimated by the results of a classifier instead of human judgments. That is, the reward is defined as the percentage of documents either labeled as relevant, or unlabeled but predicted as relevant in the newly retrieved page.

In order to reduce the dependence among arms, it is desirable that the queries are as diverse as possible. We resort to the diverse-Rocchio strategy proposed in [16] to generate new queries. Unlike the original Rocchio method [23] that utilizes all relevant documents to reformulate a query, diverse-Rocchio uses only relevant documents that are ranked lower by previous queries. That is, it tries to formulate a new query that would rank highly those documents that are relevant but not highly ranked by existing queries.

The Algorithm.

We now propose an algorithm that selects among a pool of queries in a ReQ-ReC process by treating them as bandit arms. For clarity, we list notations in Table 1.

As Algorithm 1 shows, the unlabeled document set $\mathcal{D}_q$ is initialized by retrieving the first page of results using the initial query q_0. Given $\mathcal{D}_q$, the process directly goes into the first inner loop, where a set of documents are selected by active learning strategies for user judgments. Having updated the labeled set $\mathcal{D}_l$, we train the classifier which predicts the relevance of unjudged documents $\mathcal{D}_q$. As in [16], the inner loop stops when either: (1) the performance of the classifier stabilizes, measured by changes in predictions; or (2) the limit of user judgments (L) is hit.

The outer loop then chooses a query to retrieve another page of results. First, a new query is proposed using the diverse-Rocchio approach described above. Its estimated UCB score is compared with the scores of existing queries. The query with largest score

Table 1: Notations. K, T, L are rate limits of either the search service or human effort.

$\mathcal{D}$	index of the document collection
$\mathcal{Q}$	query pool $\{\boldsymbol{q}_i\}$
q_i	the i-th query generated
K	maximum number of terms in a query
T	maximum number of API calls (pages of results per task)
L	maximum number of user judgments
$\mathcal{D}_l$	set of labeled documents
$\mathcal{D}_q$	set of unlabeled documents retrieved by query set $\mathcal{Q}$
f	classifier that predicts the relevance label of a document

Algorithm 1 ReQ-ReC by bandits

```
Input: Initial query $q_0$, index of document collection $\mathcal{D}$, number
    of terms $K$ in a query, number of API calls $T$, number of judg-
    ments $L$.
Output: A set of labeled documents $\mathcal{D}_l$ and a set of unjudged doc-
    uments in $\mathcal{D}_q$ with system predicted labels.
 1: $q \leftarrow q_0$
 2: $\mathcal{Q} \leftarrow \{q\}$
 3: $\mathcal{D}_l \leftarrow \emptyset$
 4: $\mathcal{D}_q \leftarrow retrieve\_next\_page(\mathcal{D}, q)$
 5: $t \leftarrow 1$ // record API calls
 6: repeat // outer loop
 7:   repeat // inner loop
 8:     if $|D_l| < L$ then // have budget for labeling
 9:       $\mathcal{D}_s \leftarrow select\_and\_label(\mathcal{D}_q)$ // active learning
10:       $\mathcal{D}_l \leftarrow \mathcal{D}_l \cup \mathcal{D}_s$
11:       $\mathcal{D}_q \leftarrow \mathcal{D}_q - \mathcal{D}_s$
12:     end if
13:     $train(f, \mathcal{D}_l)$
14:     $predict(f, \mathcal{D}_q)$
15:   until meet stopping criteria for inner loop
16:   if $t < T$ then // have budget for API calls
17:     $q_n \leftarrow new\_query(\mathcal{Q}, K, \mathcal{D}_q, \mathcal{D}_l)$
18:     $q \leftarrow max\_ucb(\mathcal{Q}, q_n)$
19:     $\mathcal{Q} \leftarrow \{q\} \cup \mathcal{Q}$
20:     $\mathcal{D}_q \leftarrow retrieve\_next(\mathcal{D}, q) \cup \mathcal{D}_q$
21:     $t \leftarrow t + 1$
22:   end if
23: until stopping criteria for outer loop are met
```

is chosen. If the chosen query happens to be the newly generated one, it will be added into the query pool $\mathcal{Q}$, which otherwise remains unchanged. Given the chosen query, its next page of results is retrieved and merged into $\mathcal{D}_q$.

Compared with the original ReQ-ReC process proposed in [16], there is a major difference in the termination criteria of the outer loop – it stops only when **both** of the conditions are satisfied: (1) it has already made T API calls; (2) it has already obtained L relevance judgments. Even if the user does not provide more labels, the outer loop can go on to update $\mathcal{D}_q$, thus enhancing recall. Even if no more results can be retrieved, the inner loop can continue to obtain more labels and improve the classifier, thus improving precision. This practice guarantees that we can fully utilize various types of limited resources. Below we conduct empirical experiments to demonstrate the effectiveness of query pools with bandits.

4. EXPERIMENTS

We design three simulation-based experiments to compare query pools and single combined queries. The first two use pre-calculated fixed pools of queries. One establishes a lower bound on the performance of query pools. The second improves on that using bandits to assign turns. A third experiment continuously generates new queries during the retrieval process and adds new arms.

Table 2: Basic information of data sets

	#documents	avg dl	#topics (IDs)	#qrels
MB12	15,012,766	19	59 (51-110)	69,045
MB13	243,000,000	14	60 (111-170)	71,279
20NG	18,828	225	20 categories	18,828
HARD	1,033,461	353	50 (303-689)	37,798
ClueWeb09	503,903,810	1570	50 (101-150)	64,868

* Topic 76 of MB12 has no judgment. HARD has non-consecutive topic IDs. Queries of the TREC diversity task are used for ClueWeb09.

4.1 Data sets

We select five publicly available data sets, small and large, which are commonly used in literature for a diverse set of tasks. Their basic statistics are reported in Table 2. Among them, the 20-newsgroup data set (20NG) is the smallest but fully labeled with topic categories. We include it for the purpose of understanding the behaviors of proposed methods. In particular, we intend to control the potential effect of incomplete judgments, which might be a sensitive issue for estimating the reward for bandits. The remaining four are large TREC data sets, which were used by the TREC-2005 HARD Track (HARD)[1], the TREC-2012 microblog track (MB12)[2], the TREC-2013 microblog track (MB13)[3], and the diversity task of TREC-2011 Web Track (ClueWeb09[4], category A)[5]. All these data sets are indexed and can be accessed by standard search APIs.

Following the literature [8, 27, 32, 16], we use existing TREC judgments for each query to "automate" the actual user feedback in the searches to facilitate comprehensive and fair comparisons. To deal with documents not judged by TREC, we follow the norm in the literature [27]: (1) when an algorithm (e.g., an active learner) requests the label of an unjudged document, we ignore that document and fetch the next document available, as labeling an unjudged but relevant document as irrelevant may seriously confuse a classifier; (2) when measuring the performance of a retrieved list, we treat all unjudged documents as negative.

As in Li et al. [16], documents are tokenized with Lucene's StandardAnalyzer tool and stemmed by the Krovetz stemmer [14]. Stop words are not removed. Both MB13 and ClueWeb09 provide official search APIs, implemented using the Dirichlet prior retrieval function [33]. For other data sets, we use Lucene[6] to implement the Dirichlet prior API.

Table 3: Parameters of Rocchio (α fixed as 1) for the first experiment: query pools vs. single query. Subscript s and p stand for single queries and query pools respectively.

	β_s	γ_s	β_p	γ_p
20NG	1.2	0.5	1.9	0.3
HARD	1.5	0.6	1.9	0.5
ClueWeb09	0.75	0.15	0.75	0.15

[1]http://ciir.cs.umass.edu/research/hard/
[2]http://trec.nist.gov/data/microblog2012.html
[3]https://github.com/lintool/twitter-tools/wiki/TREC-2013-Track-Guidelines
[4]http://lemurproject.org/clueweb09/
[5]http://trec.nist.gov/data/web2011.html
[6]http://lucene.apache.org/

4.2 Query Pools vs. Single Queries

The first experiment aims to validate the *potential* of query pools compared with single queries. In particular, the purpose is to estimate a *lower-bound* and a *near upper-bound* of the performance of query pools. For fair comparisons, a query pool and a single query are generated from the same set of subtopics of every retrieval task.

4.2.1 Procedure

The diversity task of the TREC-2011 Web Track annotates every search topic with subtopics, which can be directly used to construct the pool of queries. A single query is obtained by concatenating all these subtopic queries. Other data sets only provide single search topics. For every topic, we cluster all the relevant documents in the judgments using the K-means algorithm ($k = 5$) and treat them as the subtopics of the corresponding information need. To decorrelate the data before clustering, we apply the Principal Component Analysis (PCA) and use the top 50 principle components to transform the term-document TF-IDF matrix. From each cluster obtained by K-means, we generate a subtopic query that blends the relevant documents belonging to this cluster together with all non-relevant documents using Rocchio's method [23] (as if they are positive and negative feedback). Similarly, by feeding all labeled documents of a search topic to Rocchio, we obtain a single query for that topic. We limit the number of terms in the formulated query to 10.

The parameters of Rocchio are tuned to optimize the first page of retrieved results (the first search API call). For the query pool, this is the best first page returned by any of the queries in the pool. As ClueWeb09 has a restricted API limit which prevents us from exhaustive parameter searching, we set the Rocchio parameters to the recommended values in [21]). As we do not own the MB13 data set, we are unable to obtain all the labeled documents. Meanwhile, clusters obtained for MB12 queries are nearly identical. Therefore we do not use those two data sets in these experiments. The specific values of parameters are reported in Table 3.

Having obtained the queries, the retrieval process is executed as follows. The strategy for launching single queries is straightforward – we simply request the next page of results (10 per page) until we hit the limit of API calls (e.g., $T = 100$). To launch a pool of subtopic queries, we examine two strategies. The first one is referred to as the GREEDY strategy, which chooses the query whose next 10 pages (100 results) returns most relevant results. Note that by assuming omniscience, this strategy is close to an **upper-bound** of the performance of query pools. (The true upper-bound is higher and can be found by examining all possible orders to schedule the queries in the pool, which is computationally expensive.) If even this strategy is unable to outperform a single complex query, then a bandit algorithm will not help. The second strategy is the ROUND-ROBIN, which simply launches queries in circular order without looking at the returned results or the judgments. This strategy estimates a **lower-bound** of the performance of query pools. If even this lower-bound outperforms single queries, we should believe that query pools have a large potential.

Recall is a suitable evaluation metric for this experiment. All methods return nearly the same number of documents: each makes the same number of API calls; only retrieving the same document by multiple queries reduces the total number of distinct documents. Thus, precision is nearly perfectly correlated with recall.

4.2.2 Results

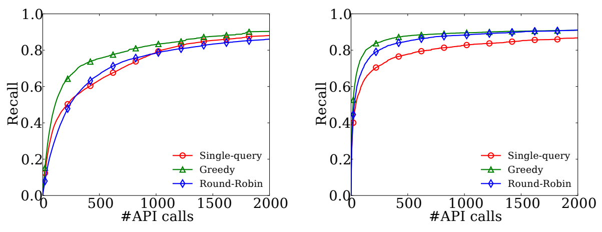
Behavior when the number of API calls is limited. Table 4 presents the recall of single queries and query pools after 100 search API calls are made, a rather limited number. The results show that query pools using the GREEDY strategy outperform single queries by a large margin, while those using the ROUND-ROBIN strategy outperform single queries on one of three data sets. This suggests that query pools have a high potential to outperform single, complex queries, but a smart way (smarter than round-robin) to choose queries in the pool is critical.

Table 4: Recall (%) at API call = 100 for the first experiment: query pools vs. single query.

	20NG	HARD	ClueWeb09
SINGLE-QUERY	38.16	61.47	50.13
ROUND-ROBIN	29.22***	**69.09*****	47.63*
GREEDY	**47.88*****	**76.68*****	**56.27****

"**(***)" means the result is significantly higher/lower than SINGLE-QUERY according to paired t-test at level 0.05(0.01).

Behavior as the number of API calls increases. We plot recall as a function of API calls in Figure 2. At the early stage, GREEDY consistently outperforms SINGLE-QUERY by a large margin. As the number of API calls increases, the advantage of GREEDY over SINGLE-QUERY is shrinking. This behavior verifies our explanation from the perspective of the information space. GREEDY prioritizes the expansion of areas covered by promising queries, thus bringing in more relevant documents earlier. As the number of allowed API calls increases, the expanded area of each query can eventually cover all relevant documents.



(a) 20NG (b) HARD

Figure 2: Recall as a function of API calls: query pools vs. single query. Results on ClueWeb09 present the same pattern.

Performance with respect to task complexity. If the information need of a retrieval task is inherently simple, a single query should be good enough. In contrast, when the information need has multiple aspects, query pools are better choice. This is already implied in Table 4, where even ROUND-ROBIN outperformed SINGLE-QUERY on the HARD data set. Here we empirically study the relationship between task complexity and retrieval performance.

To quantify task complexity, we measure the dissimilarity between generated subtopics. Intuitively, if all the subtopics of a task (topic) are similar to each other, it is likely that this topic does not have multiple aspects, and therefore is a simple task. In order to compute subtopic dissimilarity, we treat the retrieved results of each subtopic as a set. In this case, dissimilarity can be measured using the Jaccard index: $d(A, B) = 1 - J(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$, where A, B are sets. Then the task complexity is computed as the maximum of dissimilarity over every pair of subtopics belonging to one task. To make the performance across tasks comparable, we use the relative recall, which is the ratio of recall score obtained by GREEDY to that of SINGLE-QUERY.

Figure 3 presents the results, where all the three data sets are plotted in one figure. The line fitting the scattered circles suggests that the advantage of query pools is indeed more evident when the search task is more complicated.

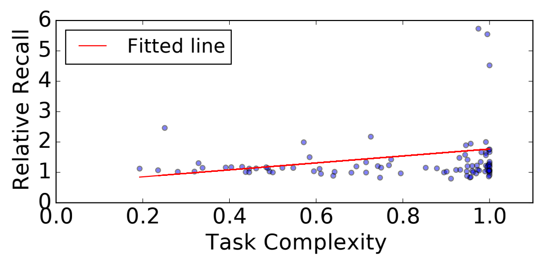


Figure 3: Task complexity vs. performance. Each circle corresponds to a retrieval task.

4.3 Bandits for Choosing Queries from a Pool

The first experiment validates the potential benefit of query pools compared with single queries. The second experiment tests whether bandits, which should do better than round-robin, perform significantly better than single queries. Since we care about the practical performance in this experiment, we no longer assume that the subtopics are given. Instead, we start from a single query and generate a set of subtopic queries based on the results obtained from the first few API calls. Note that once the subtopic queries are generated, they are fixed and no new queries will be included. We will study bandits with new queries later.

4.3.1 Procedure

We generate subtopics by first applying active feedback [27] except that we only use the results retrieved from the first few API calls. As the first experiment shows, query pools are more powerful for complex tasks. To filter out simple tasks, here we use a heuristic – a task will be dropped if active feedback cannot obtain a certain number of relevant documents $\theta_p = 10$ within 20 API calls. This suggests that there are not enough diverse cluster "centroids" in these simple tasks. The number of "complex" tasks remaining are shown in the first column of Table 5.

Table 5: Number of tasks and Rocchio parameters (α fixed as 1) to evaluate bandits for choosing queries from a pool. Subscript s and b stands for SINGLE-QUERY and BANDIT. Iterative RF has the same first reformulated query as SINGLE-QUERY and they share the same parameter values.

	#tasks	β_s	γ_s	β_b	γ_b
20NG	19	0.9	0.3	1.5	0.5
HARD	33	1.0	0.3	1.5	0.6
ClueWeb09	31	0.75	0.15	0.75	0.15

The labeled documents acquired by active feedback are clustered and one query per cluster is generated by Rocchio's method [23]. Again, parameters are tuned to optimize the first page of results. We set k to 2 in the K-means algorithm this time, as there are fewer relevant documents obtained. The single query is also obtained by feeding all retrieved, labeled documents to Rocchio.

Given the two generated queries, the bandit algorithm is applied to choose which query to explore next. Parameters c and τ in Equation 6 are respectively set to 0.1 and 20 across all data sets.

Since the bandit further requests relevance feedback to estimate the reward, to make the comparison fair we add another baseline – iterative relevance feedback (ITERATIVE RF). It uses the same user judgments as the bandit algorithm, but instead updates the query in every iteration using Rocchio's method. Throughout the process, only the newest query is active.

4.3.2 Results

Table 6 summarizes the results. The results show that the bandit algorithm can effectively explore the two subtopic queries and exploit the better one, retrieving more relevant results overall. Though ITERATIVE RF demands the same number of labeled documents as BANDIT per iteration, it utilizes them less effectively by updating a single query. Note that the results might not be comparable to the ones presented in Table 4, as there are positive documents already retrieved by active feedback at API call = 0.

Table 6: Recall (%) at API call = 100 for the second experiment: bandits order queries from a pool more effectively.

	20NG	HARD	ClueWeb09
Iterative RF	19.44***	39.33***	39.27***
SINGLE-QUERY	40.41	64.70	46.26
BANDIT	**43.42***	**68.85****	**48.61****

"**(***)" means the result is significant over SINGLE-QUERY according to paired t-test at level 0.05(0.01).

We plot recall as a function of API calls in Figure 4. In ClueWeb09 and HARD, the BANDIT algorithm consistently outperforms SINGLE-QUERY. As for 20NG, BANDIT shows only some slight advantage in the beginning, and dominates by a large margin at later stages. This might result from the unavailability of all labeled relevant documents, which is only accessible in the first set of experiments. Without them, a single query generated from documents retrieved by active feedback might under-represent some subtopics. Therefore in terms of the information space, the query could be located far away from those minor subtopics, and thus takes more API calls to expand its area to cover these subtopics.

4.4 Bandits with New Queries

Having shown that the bandit algorithm can effectively assign turns to a fixed pool of queries, we test whether it can handle new queries generated on the fly, in an interactive retrieval process.

4.4.1 Metrics

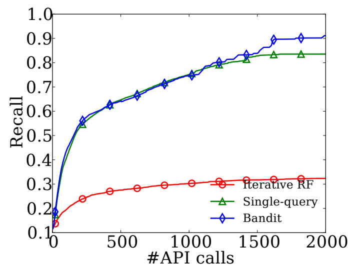
We use two standard retrieval metrics that emphasize both recall and precision – mean average precision (MAP) [21] and R-precision (R-Prec) [21]. R-precision measures the precision at the R-th position for a query with R relevant judgments. The R-th position is where precision equals recall. This makes it a good metric that takes into account both precision and recall. These metrics are also used in [16]. For each query, only the top 1,000 documents are considered when computing the metrics.

When measuring retrieval performance, the documents already judged by the user in an interactive search process are included, and those judged as relevant are put on the top of the ranked list. As reasoned in [16], it is not fair to punish a process that does a good job of sending more relevant documents to the user for judgments.

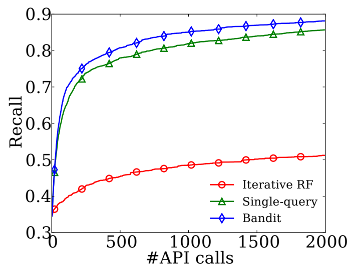
4.4.2 Methods

The baselines include ITERATIVE RF and the best-performing configuration of the ReQ-ReC framework reported in [16], simply referred to as REQ-REC here.

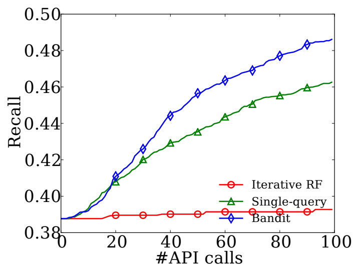
Since the new process considers rate limits, we accordingly make adaption to baselines, ensuring that the comparisons are fair. First, as baselines cannot revisit queries once they have been updated, we retrieve P pages all at once for each query, where P is tuned to the best value for each baseline. Second, the stop criteria for the outer loop are changed to the ones described in 3.3.2. That is, the entire process terminates only when we reach the limit on both the number of labeled documents and the number of API calls.



(a) 20NG



(b) HARD



(c) ClueWeb09

Figure 4: Results for the second experiment: bandits for choosing queries from a pool

We do not remove simple tasks as we did in the second experiment, as BANDIT automatically decides when to generate a new arm. In nearly all cases, it indeed generated more than one arm.

4.4.3 *Parameters*

For each baseline on each data set, parameter P is tuned to maximize the mean average precision. MB13 and ClueWeb09 are exceptions where this tuning is not possible due to rate limits of the official APIs. We set the P to be the mode of the best P on the other three data sets. The values of P, as well as tuned parameters for Rocchio, are shown in Table 7.

We do not further tune the parameters of BANDIT even though they may not be the optimal values. For c, τ, $\hat{c}$ in Equation 6 and 7, they are respectively set to 0.1, 20, and 0.1 across all data sets.

In all the methods, we use the default parameter of SVM ($c = 1$). With regard to the resource limit, we set the number of terms in a query $K = 5$, number of API calls $T = 100$, and number of labeled documents $L = 20$. A relaxation of these constraints will be studied in Section 4.4.5.

4.4.4 *Overall Performance*

Table 8 summarizes the performance of all included methods. Statistical significance of the results are provided by comparing to the baseline, REQ-REC. In general, BANDIT is the best performing method across all data sets.

Table 7: The number of pages to retrieve for each query (P) for baselines and parameters of Rocchio (α fixed as 1). Subscript i, r and b stands for method ITERATIVE RF, REQ-REC and BANDIT respectively.

	P_i	P_r	β_i	γ_i	β_r	γ_r	β_b	γ_b
MB12	30	40	1.5	0.3	1.7	1.0	1.8	0.1
MB13	30	40	0.75	0.15	0.75	0.15	0.75	0.15
20NG	30	20	0.9	0.5	0.9	0.4	1.0	0.4
HARD	30	40	1.8	0.2	0.9	0.4	1.9	0.2
ClueWeb09	30	40	0.75	0.15	0.75	0.15	0.75	0.15

When comparing REQ-REC with ITERATIVE RF, we obtain mixed results. The problem with REQ-REC is caused by the lack of enough training examples for the classifier, which fails to identify relevant documents and rank them on the top. This problem has been mentioned in [16] too. Though BANDIT suffers from the same problem, it retrieves more relevant documents by managing queries smartly, alleviating the problem of relevance classification. This will be clearer in the next subsection when we analyze the results by increasing the number of labeled documents.

4.4.5 *Relaxation of constraints*

In order to study the effects of different types of constraints, we relax one of the factors at a time, while keeping the other two fixed. As the result of R-prec is strongly correlated with MAP, to conserve space we only show the performance of each method by MAP.

Table 9: MAP (%) as the number of terms in a query increases.

Data set	Method	5	10	20
MB12	Iterative RF	27.19*	30.25***	31.87***
	ReQ-ReC	25.08	25.12	25.31
	Bandit	**30.97*****	**31.68*****	**32.58*****
MB13	Iterative RF	24.18	29.68***	30.34***
	ReQ-ReC	25.49	25.50	25.41
	Bandit	**30.19*****	**32.43*****	**33.28*****
20NG	Iterative RF	10.02	10.49	11.10
	ReQ-ReC	11.81	11.78	11.88
	Bandit	**18.92*****	**19.00*****	**18.56*****
HARD	Iterative RF	18.47***	21.02**	23.05
	ReQ-ReC	24.35	24.29	24.37
	Bandit	**26.22***	**26.38***	**26.35***
ClueWeb09	Iterative RF	8.45***	9.62***	9.89***
	ReQ-ReC	13.28	13.40	13.41
	Bandit	**14.86***	**14.84***	**14.81**

Table 9 shows the performance when increasing the number of terms in a query. As the number of allowed terms is increased, the performance of ITERATIVE RF is enhanced significantly, while BANDIT and REQ-REC have no evident change. This is attributed to the classifier maintaining precision in the inner loop of the ReQ-ReC framework. As a result, the query generator of BANDIT and REQ-REC could focus on maximizing recall by exploring diverse sets of term, which is feasible even with strict query length constraints. In contrast, without the help of a classifier, the queries generated by ITERATIVE RF have to account for both precision and recall, a difficult task when the number of query terms is limited.

Table 10 suggests that increasing the number of labeled documents in general benefits REQ-REC and BANDIT greatly. This benefit can be explained from two perspectives. More labeled documents leads to: 1) a more robust classifier to judge document relevance; and 2) generation of more diverse queries. When labeled documents are few, even though BANDIT and REQ-REC can retrieve more relevant documents than ITERATIVE RF, they are not recognized by the classifier. With more labeling efforts, the two methods outperform ITERATIVE RF by a large margin. The difference between BANDIT and REQ-REC tend to shrink on MB12 and MB13. This could be due to more labeled documents contributing to increased quality of all queries in the pool. Choosing any of the queries to submit can already result in good performance, even if this is not managed by bandits.

Table 11 presents the results when increasing the number of API

Table 8: Retrieval performance (%) with query terms $K = 5$, API calls $T = 100$, labeled documents $L = 20$.

	MB12		MB13		20NG		HARD		ClueWeb09	
	R-prec	MAP	R-prec	MAP	R-prec	MAP	R-prec	MAP	R-prec	MAP
Iterative RF	31.29	27.19*	30.59	24.18	13.84**	10.02	24.12***	18.47***	13.28***	8.45***
ReQ-ReC	29.37	25.08	30.94	25.49	17.67	11.81	27.93	24.35	17.97	13.28
Bandit	32.51**	30.97***	34.89***	30.19***	28.84***	18.92***	29.83*	26.22*	19.5*	14.86*

"*(**, ***)" means the result is significant over ReQ-ReC according to paired t-test at level 0.1(0.05, 0.01).

Table 10: MAP (%) as labeled documents increase.

Data set	Method	20	30	60	100
MB12	Iterative RF	27.19*	26.37**	30.92***	32.12***
	ReQ-ReC	25.08	29.30	39.45	46.53
	Bandit	30.97***	35.49***	41.85**	49.68**
MB13	Iterative RF	24.18	24.50***	27.42***	28.39***
	ReQ-ReC	25.49	28.79	35.98	39.98
	Bandit	30.19***	32.39***	38.83***	41.54*
20NG	Iterative RF	10.02	10.36*	11.57**	12.80***
	ReQ-ReC	11.81	12.66	16.69	18.22
	Bandit	18.92***	20.45***	23.22***	25.43***
HARD	Iterative RF	18.47***	17.18***	19.51***	20.28***
	ReQ-ReC	24.35	28.22	37.47	42.86
	Bandit	26.22*	**29.37**	39.12**	47.16***
ClueWeb09	Iterative RF	8.45***	8.05***	8.65***	8.65***
	ReQ-ReC	13.28	14.90	20.42	25.35
	Bandit	14.86*	**15.32**	22.65*	29.56***

Table 11: MAP (%) as the number of API calls increases.

Data set	Method	100	500	1000
MB12	Iterative RF	27.19*	27.19***	27.19***
	ReQ-ReC	25.08	18.65	17.49
	Bandit	30.97***	28.35***	35.38***
MB13	Iterative RF	24.18	26.18***	26.18***
	ReQ-ReC	25.49	21.32	20.31
	Bandit	30.19***	28.20***	26.84***
20NG	Iterative RF	10.02	10.02	10.02
	ReQ-ReC	11.81	10.53	10.5
	Bandit	18.92***	18.75***	21.49***
HARD	Iterative RF	18.47***	18.47**	18.47***
	ReQ-ReC	24.35	15.37	14.06
	Bandit	26.22*	19.80***	33.25***
ClueWeb09	Iterative RF	8.45***	8.45*	8.45*
	ReQ-ReC	13.28	12.09	11.87
	Bandit	14.86*	**13.91**	**12.54**

calls. No change of performance is observed for ITERATIVE RF, as the query will not be changed after running out of labeling budget. As a result, ITERATIVE RF always outputs the same ranked list. In many cases BANDIT and REQ-REC deteriorate, due to the classifier's errors on newly retrieved documents when no more new labels are available to correct the classifier. As observed from Table 10, the classifier becomes more robust as the labeling effort increases. Therefore, to fully utilize the increased API resource, the number of judgments has to be increased accordingly. Interestingly, on three of the data sets, BANDIT obtains a big gain when the number of API calls is increased to 1,000. By smartly managing a pool of queries and revisiting promising ones, BANDIT can still bring in more relevant documents, even with a biased classifier.

We also study τ that controls the size of window in sliding-window UCB [6]. Overall, the performance is fairly robust over different values of τ. When the window size is extremely small ($\tau = 5$), UCB does not have enough history to correctly estimate rewards. The problem is alleviated as we expand the window. Performance begins to drop when τ is greater than 10. It is beneficial to forget rewards obtained earlier, as the number of relevant documents per page generally decreases when we retrieve more pages.

4.5 Result Reranking as a Post-processing

Query pools aim at retrieving more relevant documents from a black-box search engine. The retrieved results can be fed to any reranking method, including any algorithm developed for session search or result diversification. These methods serve as a post-processor to reorder retrieved documents and potentially improve the effectiveness of ranking. In this subsection, we use xQuAD [24], a popular result diversification algorithm, to examine the benefit of such post-processing. xQuAD considers different aspects of the information need by exploiting the ranking score of sub-queries. We directly use the set of queries generated in our system with uniform weights as the sub-queries for xQuAD, and tune the parameters to their best. Table 12 and 13 show results of applying xQuAD to rerank retrieved documents for the second and third experiments.

Table 12: MAP (%) at API call = 100 for the second experiment: bandits for choosing queries from a pool.

	20NG	HARD	ClueWeb09
SINGLE-QUERY-original	14.89	21.68	14.35
SINGLE-QUERY-xQuAD	16.31**	23.24**	14.64
Bandit-original	18.81	23.76	16.34
Bandit-xQuAD	20.26**	25.21**	16.73

"**" means the reranking of xQuAD is significant over the original ranking according to paired t-test at level 0.05.

Table 13: MAP (%) at API call = 100 for the third experiment: bandits with new queries.

	MB12	MB13	20NG	HARD	ClueWeb09
Iterative-RF-original	27.19	24.18	10.02	18.47	8.45
Iterative-RF-xQuAD	28.34*	24.33	10.94	19.58	20.56***
ReQ-ReC-original	25.08	25.49	11.81	24.35	13.28
ReQ-ReC-xQuAD	27.69**	24.9	12.47	24.92	23.11***
Bandit-original	30.97	30.19	18.92	26.22	14.86
Bandit-xQuAD	32.68*	29.82	18.10	28.32**	27.81***

"**" means the reranking of xQuAD is significant over the original ranking according to paired t-test at level 0.05.

The overall performance indicates that xQuAD can improve the original ranking in many cases, while in other cases the performance stays very close. This suggests that there is no harm to add a re-ranker, which often benefits. The most interesting case is ClueWeb09 in Table 13, where the reranked MAP is around twice as high as the original. We suspect that web pages are noisier than other documents and it is easy for a classifier to misclassify due to noises. By considering subtopics in the ranking function, xQuAD alleviates this problem and promotes relevant documents to the top.

Numbers in Table 12 and 13 are not comparable, as experiments reported in Table 12 utilized active feedback to accumulate positive documents before we start counting API calls. Moreover, BANDIT in Table 12 requires far more labels than that in Table 13.

5. CONCLUSION

We consider a scenario where users have to rely on rate limited search services to acquire documents pertaining to an information need. Traditional approaches do not address this problem well when a user has a complex information need composed of multiple aspects. These approaches keep a single active query for document retrieval. This single query might either be fixed or updated by techniques like relevance feedback. We propose a new retrieval paradigm where a pool of queries are kept active simultaneously, and a bandit algorithm is used to determine which query to retrieve another page of results for. The empirical results demonstrate the advantage of query pools over solo, combined queries. The improvement of performance is significant both when the query aspects are known in advance and when new queries are generated in an interactive retrieval process. The new search paradigm can be launched as a general interface between a user and any black-box search API. The retrieved documents can be further reranked by any post-processing algorithm to improve the ranking performance. It is worth noting that throughout the study we considered the search engine as a black box. When one owns the search systems, there is much more flexibility to improve both the algorithm (e.g., to use query and clickthrough logs) and the evaluation (e.g., to identify diverse subtopic queries from the query log instead of reverse-engineering the TREC topics).

Acknowledgment

The authors thank Kevyn Collins-Thompson, ChengXiang Zhai, and reviewers for their useful comments. This work is partially supported by the National Science Foundation under grant numbers IIS-0968489 and IIS-1054199 and by the National Institutes of Health under grant number NLM 2R01LM010681-05.

6. REFERENCES

[1] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *Proc. of WSDM*, 2009.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 2002.

[3] D. Bouneffouf, A. Bouzeghoub, and A. L. Gançarski. Contextual bandits for context-based information retrieval. In *Neural Information Processing*. Springer, 2013.

[4] J. Carbonell and J. Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proc. of SIGIR*, 1998.

[5] B. Carterette, E. Kanoulas, M. Hall, and P. Clough. Overview of the trec 2013 session track.

[6] A. Garivier and E. Moulines. *On Upper-Confidence Bound Policies for Switching Bandit Problems*. 2011.

[7] D. Guan, H. Yang, and N. Goharian. Effective structured query formulation for session search. Technical report, DTIC Document, 2012.

[8] D. Harman. Relevance feedback revisited. In *Proc. of SIGIR*, 1992.

[9] J. He, V. Hollink, and A. de Vries. Combining implicit and explicit topic representations for result diversification. In *Proc. of SIGIR*, 2012.

[10] C.-C. Hsieh, J. Neufeld, T. King, and J. Cho. Efficient approximate thompson sampling for search query recommendation. 2015.

[11] J. Jiang, S. Han, J. Wu, and D. He. Pitt at trec 2011 session track. In *TREC*, 2011.

[12] X. Jin, M. Sloan, and J. Wang. Interactive exploratory search for multi page search results. In *Proc. of WWW*, 2013.

[13] K. Jones and V. Rijsbergen. *Report on the Need for and Provision of an Ideal Information Retrieval Test Collection*. 1975.

[14] R. Krovetz. Viewing morphology as an inference process. In *Proc. of SIGIR*, 1993.

[15] T. L. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 1985.

[16] C. Li, Y. Wang, P. Resnick, and Q. Mei. Req-rec: High recall retrieval with query pooling and interactive classification. In *Proc. of SIGIR*, 2014.

[17] L. Li, W. Chu, J. Langford, and X. Wang. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proc. of WSDM*, 2011.

[18] D. E. Losada, J. Parapar, and A. Barreiro. Feeling lucky? multi-armed bandits for ordering judgements in pooling-based evaluation. In *31st Symposium on Applied Computing*, 2016.

[19] J. Luo, S. Zhang, X. Dong, and H. Yang. Designing states, actions, and rewards for using pomdp in session search. In *Advances in Information Retrieval*. 2015.

[20] J. Luo, S. Zhang, and H. Yang. Win-win search: Dual-agent stochastic game in session search. In *Proc. of SIGIR*, 2014.

[21] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*. Cambridge University Press Cambridge, 2008.

[22] F. Radlinski and S. Dumais. Improving personalized web search using result diversification. In *Proc. of SIGIR*, 2006.

[23] J. J. Rocchio. Relevance feedback in information retrieval. 1971.

[24] R. L. Santos, C. Macdonald, and I. Ounis. Exploiting query reformulations for web search result diversification. In *Proc. of WWW*, 2010.

[25] B. Settles. Active learning literature survey. *University of Wisconsin, Madison*, 2010.

[26] X. Shen, B. Tan, and C. Zhai. Context-sensitive information retrieval using implicit feedback. In *Proc. of SIGIR*, 2005.

[27] X. Shen and C. Zhai. Active feedback in ad hoc information retrieval. In *Proc. of SIGIR*, 2005.

[28] M. Sloan and J. Wang. Dynamical information retrieval modelling: a portfolio-armed bandit machine approach. In *Proc. of WWW*, 2012.

[29] M. Sloan and J. Wang. Dynamic information retrieval: Theoretical framework and application. In *Proc. of ICTIR*, 2015.

[30] E. J. Sondik. The optimal control of partially observable markov processes over the infinite horizon: Discounted costs. *Operations Research*, 1978.

[31] S. Srinivasan, E. Talvitie, M. Bowling, and C. Szepesvári. Improving exploration in uct using local manifolds. In *Proc. of AAAI*, 2015.

[32] X. Wang, H. Fang, and C. Zhai. A study of methods for negative relevance feedback. In *Proc. of SIGIR*, 2008.

[33] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proc. of SIGIR*, 2001.

[34] C. X. Zhai, W. W. Cohen, and J. Lafferty. Beyond independent relevance: methods and evaluation metrics for subtopic retrieval. In *Proc. of SIGIR*, 2003.