Orthogonal Query Recommendation

Hossein Vahabi *
Microblr
London, UK
pvahabi@microblr.com

Margareta Ackerman
Caltech
Pasadena, California, USA
mackerma@caltech.edu

David Loker
University of Waterloo
Waterloo, Canada
dloker@cs.uwaterloo.ca

Ricardo Baeza-Yates
Yahoo! Research Labs
Barcelona, Spain
rbaeza@acm.org

Alejandro Lopez-Ortiz
University of Waterloo
Waterloo, Canada
alopez-o@uwaterloo.ca

ABSTRACT

One important challenge of current search engines is to satisfy the users' needs when they provide a poorly formulated query. When the pages matching the user's original keywords are judged to be unsatisfactory, query recommendation techniques are used to propose alternative queries and alter the result set. These techniques search for queries that are *semantically similar* to the user's original query, often searching for keywords that are similar to the keywords given by the user. However, when the original query is sufficiently ill-posed, the user's informational need is best met using entirely different keywords, and a substantially different query may be necessary.

We propose a novel approach that is not based on the keywords of the original query. We intentionally seek out *orthogonal queries*, which are related queries that have (almost) no common terms with the user's query. This allows an orthogonal query to satisfy the user's informational need when small perturbations of the original keyword set are insufficient. By using this technique to generate query recommendations, we outperform several known approaches, being the best for long tail queries.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Query formulation*

General Terms

Algorithms, Experimentation, Measurement

Keywords

Query Recommendation, Knowledge Extraction

*This work was partially done at CNR,Pisa, Italy.

RecSys'13, October 12–16, 2013, Hong Kong, China.

http://dx.doi.org/10.1145/2507157.2507159.

1. INTRODUCTION

Over the last seventeen years there has been enormous progress on Web search. Starting from the text-based ranking algorithms of 1994, we now have complex ranking algorithms that use hundreds of features and many functionalities based on usage data, such as spelling correction, query completion and query recommendation. Due to these advances, search engines usually satisfy a user's informational need on well-formulated queries. However, an important remaining challenge is to satisfy the users' informational need when they provide vague, poorly formulated or long tail queries.

When the pages matching the user's original keywords are judged to be unsatisfactory, query recommendation algorithms are a common method for helping users find the information they need. These algorithms aim to provide queries that are, at least somewhat, *similar* to the original [2].

Another approach is to use query expansion [2], where the goal is to find keywords that, while syntactically different to varying degrees, have the same semantics as the keywords in the original query. Query recommendation can be seen as query expansion where we limit the query universe to those queries that have been previously input by some user.

Using these approaches, the new result set is a perturbation of the original result set. When the query is sufficiently well composed a small perturbation would be sufficient; in those cases, there is a highly ranked page relevant to the user's needs that appears in the result set of the new query, whereas the result set of the original query did not contain such results. Traditional approaches to query recommendation play an important and necessary role in helping correct queries that require minor adjustment.

However, users cannot always pick the most appropriate keywords. This is not surprising, because some queries are launched precisely because users wish to learn a subject with which they have little familiarity. When the query is sufficiently ill-posed, the user's informational need is best met using entirely different keywords, and a small perturbation of the original result set is bound to fail. Interestingly, in this case, the higher the query similarity used for the perturbation, the less likely that the recommendation would succeed.

We propose a new approach that does not directly rely on the original keyword set, and indeed does not rely on queries that are syntactically similar to the original. We intention-

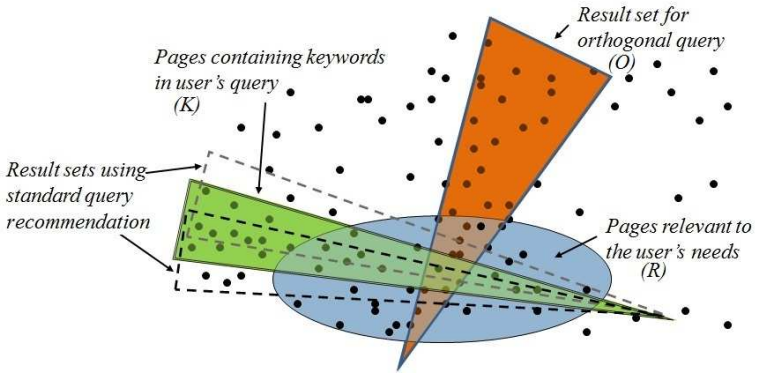

Figure 1: A graphical illustration of the difference between traditional query recommendation and orthogonal query recommendation. The dots represent webpages. The oval represents the set of pages that are relevant to the user's needs. The pages containing the user's keywords are represented in the green, nearly-horizontal stab. Results sets using traditional query recommendation are shown using the dashed stabs. An orthogonal result set is represented using the orange stab, appearing perpendicular to the original result set.

ally seek out *orthogonal queries*, which are related queries that have *no common terms* with the user's query. That is, orthogonal queries contain terms that are *syntactically* different from the terms in the original query, but that are still *semantically similar*. Hence, they provide insightful alternative interpretations that are not reachable by using small variations on the original keyword set.

Orthogonal query recommendation is complementary to traditional query recommendation, each technique may succeed when the other fails. Traditional approaches explore adjacent meanings of the user's query, whereas orthogonal query recommendation considers relevant interpretations that are more distant. See Figure 1 for an illustration of our main idea. Orthogonal queries tap into the space of relevant pages in a radically different way than is possible through traditional query recommendation techniques, allowing them to detect high quality pages that cannot be found by using previous techniques. Observe that if the original query is sufficiently ill posed, no small perturbation will succeed in capturing high value pages. In addition, an orthogonal query can access the user's informational need while consisting of keywords that are mostly distinct from those in the original query.

The challenge is to find orthogonal queries in a computationally efficient manner that proves useful in practice. We find orthogonal queries by taking advantage of the vast amounts of data that search engines collect, finding queries with low result similarity in an *ad-hoc* query recommendation answer cache. The use of an answer cache enables us to take advantage of temporal locality, and hence the query recommendations automatically reflect current events and trends, thus further increasing the likelihood that the user's informational need is met.

We illustrate the effectiveness of this approach by proposing a query recommendation method derived from these observations, and demonstrate its effectiveness on a large query log against other existing query recommendation techniques. As a result of our evaluation, to the best of our knowledge, we present the most extensive comparison of query recommendation algorithms.

Finally, we perform a user study using the standard TREC Web diversification task test bed. We compare orthogonal query recommendation against six previous approaches, including several that are substantially more computationally extensive. Orthogonal query recommendation outperforms all previous methods. In addition, our method outperforms previous approaches by a substantial margin on queries that were not previously seen.

The rest of this paper is organized as follows. In Section 2, we discuss prior approaches to query recommendation. In Section 3, we introduce our notion of orthogonal queries in a formal manner and we discuss how we compute those queries using a particular notion of query similarity. In Section 4, we evaluate the effectiveness of our query recommendation algorithm with respect to others and conclude in Section 5.

2. PREVIOUS WORK

Query recommendation algorithms address the problem of recommending good queries to Web search engine users, and thus the solutions and evaluation metrics are tailored to the Web search domain. Recently, many different approaches have arisen to solve this problem, but they all share a common element: the exploitation of usage information recorded in query logs [13].

One subset of these recommendation algorithms use clustering to determine groups of similar queries that lead users to related documents [14, 1]. In the case of [1], the recommended queries might have completely different terms to the original query, like the algorithm proposed in this paper, as they use term similarity over of clicked answers (and not the queries). Additional approaches include exploiting chains of queries stored in query logs [10], or improving the retrieval accuracy of difficult queries [11].

Baeza-Yates and Tiberi [3] exploit click-through data as a way to provide recommendations. The method is based on the concept of Cover Graph (CG). A CG is a bipartite graph of queries and URLs, where a query q and an URL u are connected if a user issued q and clicked on u that was an answer for the query.

Another approach, similar to the previous one, called user frequency-inverse query frequency (UF-IQF), was introduced by Deng *et al.* [9] and is based on entropy models. The same year, Silvestri *et al.* [7] introduced a new recommendation algorithm: *search short cuts* (SC).

Zaiane and Strilets [15] exploit query traces to find similar queries (SQ), the queries with similar terms and results are scored higher,which is the opposite of our method.

The last four algorithms described above (CG, UF-IQF, SC, SQ) are used as baselines in our experimental comparison. Finally, as part of our user study, we compare our orthogonal query approach with two other graph based query recommendation algorithms that are not that efficient: Query Flow Graph (QFG) [5] and *Term-Query Graph* (TQG) [6].

Our method differs from previous approaches in that we deliberately seek out queries that are only *slightly* similar to the user's original query. In this way, we avoid the pitfall of recommending queries that are simple reformulations of the original, as this would do very little to bring the user closer to their informational need if the original query was poorly formed or has few results, as is the case with long tail queries.

3. ORTHOGONAL QUERIES

In our model, the objective of a search engine is to retrieve at least one highly ranked page that is relevant to the user's needs. The purpose of an orthogonal query is to satisfy the user's information need when they are not met by the original results.

Formally, let $\mathcal{R}$ denote the set of web-pages that are relevant to the user's needs. Let $\mathcal{K}$ denote the set of pages that contain the keywords comprising the user's query. Search engines rely on the existence of some highly ranked pages in $\mathcal{R} \cap \mathcal{K}$, since these would be the top results returned to the user.

As such, the main limitation of the current approach to searching is its restricted capacity to access pages in $\mathcal{R}$. We introduce orthogonal queries as a means of accessing $\mathcal{R}$ in a manner that is not as dependent on the particular keyword choices made by the user.

We refer to $\mathcal{K}$ as a *stab* of $\mathcal{R}$. An *orthogonal stab* is a set $\mathcal{O}$ such that $\mathcal{O} \cap \mathcal{K}$ is small. In particular, we are interested in orthogonal stabs so that $\mathcal{R} \cap \mathcal{O}$ contains some highly ranked pages. See Figure 1 for an illustration. *Orthogonal results* denote pages in $\mathcal{O}$ that do no occur in $\mathcal{K}$.

Orthogonal queries and their results may be useful when $\mathcal{R} \cap \mathcal{K}$ is unsatisfactory; for instance when $\mathcal{R} \cap \mathcal{K}$ does not contain enough highly ranked pages. Orthogonal query recommendation is also useful when the top results in $\mathcal{R} \cap \mathcal{K}$ address the same interpretation of the user's query, allowing orthogonal queries to capture alternative interpretations (or in other words, diversify the results). Orthogonal queries and their results may satisfy the user information needs on poorly formulated queries, by going beyond the scope of the provided keywords. These results are also able to provide relevant information that is entirely new to the user, where the user could not have searched for it directly.

To characterize orthogonal queries we use a similarity measure based in the query terms. Let the *term overlap* between queries p and q be

$$termOverlap(p,q) = \frac{|terms(p) \cap terms(q)|}{|terms(p) \cup terms(q)|}.$$

Query 1	Query 2	T.O.	R.O.
european+rabbit	*European rabbit*	1	0.575
lyrics office space	*office space lyrics*	1	0.4084
car-price	*bluebook cars*	0	0.06
discount travel	*cheap airfares*	0	0.105
Daisy Duke	*"catherine bach"*	0	0.02

Table 2: Examples of query pairs and their Term Overlap (T.O.) and Result Overlap (R.O.) scores.

Using this similarity measure, we can define initially orthogonal queries as those having *termOverlap* equal to zero. However, we will relax this definition later and allow orthogonal queries to have very low *termOverlap*, i.e. we should call them quasi-orthogonal.

3.1 Interesting Orthogonal Queries

Our orthogonal query recommendation technique relies on a measure of similarity that goes beyond keyword comparisons, and is at the same time computationally efficient so that the similarity score can be computed in real-time.

Let $resultSet(p)$ denote the set of URLs returned by a search engine on query p. The *result overlap* between queries p and q is,

$$resultOverlap(p,q) = \frac{|resultSet(p) \cap resultSet(q)|}{|resultSet(p) \cup resultSet(q)|}.$$

See, for example, Balfe *et al.* [4].

We found that queries with a large result overlap score yield results that are similar to the original query's results, and thus do not address alternative interpretations. We use the result overlap score to further determine when a query is orthogonal, and build a set of interesting orthogonal queries to be used as recommendations.

In order to find a range of result overlap that leads to interesting orthogonal queries, we compare result overlap with term overlap. We first provide a high level description of the relationship between term overlap and result overlap, and discuss later how this relationship enables us to identify a range of result overlap that leads to orthogonal queries.

Very high values of result overlap tend to indicate that the queries are composed of similar terms. The most similar queries are slight syntactic variants composed of the *same* terms. For instance, the queries *european+rabbit* and *European rabbit* have result overlap 0.575. As our algorithm compares the top 100 results from both queries, a result overlap score of 0.575 indicates that 73 of the top 100 results match. Queries that are word permutations of each other, as in *lyrics office space* and *office space lyrics* also have a high result overlap score, in this case 0.4084. Many other queries with high result overlap score often have significant overlap in their term bags.

When both the result overlap and term overlap scores are high, incorporating highly ranked results from such a query into the original result set does not significantly alter the original result set. Hence, to find pages that satisfy the needs of users when their informational needs are not met by the original highly ranked results, we look for similar queries (according to the result overlap score) that include entirely different keywords. Indeed, the most interesting results occur at a low range of result overlap. Surprisingly, we

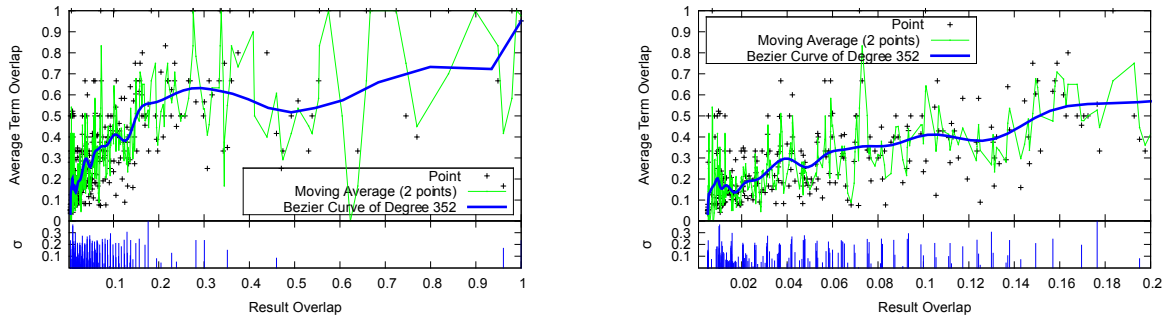

Figure 2: Result overlap versus average term overlap in a query log of $5,000$ queries. At the right we show the result overlap values in the range $(0, 1]$, while at the left the result overlap values in the range $(0, 0.2]$. The lines represents the moving average of period two, and the Bézier curve of degree 352. We also show the standard deviation σ. The graphs show a positive correlation between result and term overlap.

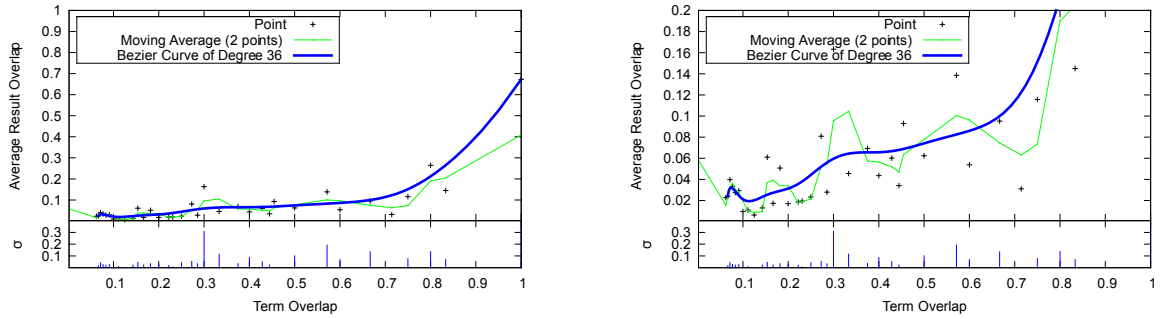

Figure 3: Term overlap versus average result overlap in a query log of $5,000$ queries. At the right we show the average result overlap values in the range $(0, 1]$, while at the left the values in the range $(0, 0.2]$. The lines represents the moving average of period two, and the Bézier curve of degree 36. We also show the standard deviation σ. The graphs show a positive correlation between results and term overlap.

did not find an instance where two queries have result overlap beyond score 0.01 and yet are semantically dissimilar.

The result overlap measure of query similarity can detect semantic similarity when there are *no common* terms, without the use of complicated natural language processing techniques. For example, the queries `students with reading difficulties` and `dyslexia help` have result overlap 0.0102, and `car-price` and `bluebook cars` have a 0.06 result overlap. A surprising relationship was caught in the comparison of the queries `Daisy Duke` and `"Catherine Bach"` with a result overlap value of 0.02. Further investigation revealed that Catherine Bach played character Daisy Duke in The Dukes of Hazard.

3.2 Finding Interesting Orthogonal Queries

Now we perform a more formal comparison of result overlap with term overlap in order to identify the most appropriate range of result overlap for finding orthogonal queries. For this we compute the result overlap and term overlap scores for each distinct pair of queries in a query log of $5,000$ entries. We computed term overlap while ignoring very common stop words (such as "a" and "the"), otherwise many dissimilar pairs of queries would have high term overlap. In addition, we reduce all letters to lower case, and treat words as sequences of alpha-numerical characters (i.e., then *european+rabbit* and *European rabbit* have term overlap 1.)

The Pearson correlation coefficient between result overlap and average term overlap is 0.567(352 data points), indicating a positive correlation. Figure 2 shows result overlap values in the range $(0, 1]$ and $(0, 0.2]$ with the corresponding average term overlap values. The lines in Figure 2 represent a moving average with a period of two, and a Bézier curve of degree n. We also report the standard deviation of term overlap for each result overlap. Since we were using a real query log, it is not surprising that the number of queries decreases as result overlap increases.

We obtain further evidence of the positive correlation between the two measures of similarity by looking at term overlap versus average result overlap. The Pearson correlation coefficient between them reveals a strong positive correlation of 0.686 (36 data points). Figure 3 shows term overlap values with the corresponding average result overlap values in the range $(0, 1]$ and $(0, 0.2]$, similarly to the previous figure.

We would like to identify a range of result overlap where term overlap is low, as most queries have very few terms. In a study of 285 million Internet users, we found that over 87% of search queries consist of 4 or fewer terms [12]. A term overlap score below 0.333, on two queries of length at most 4, indicates that the queries overlap on at most one term. With the goal of introducing as little noise as possible, and only presenting those results that are most likely to be orthogonal to, and not similar to, the results of the original query, we want to avoid having large term overlap. To this

end, we set a threshold so that we expect to have at most one overlapping term.

Taking into account the size of the Web, and thus the number of possibilities of the first 100 results, a non-zero result overlap is actually very meaningful. Indeed, the probability of a false match can be estimated in the range of 10^{-5} to 10^{-9}. We found that in practice a non-zero result overlap score in most cases represents some semantic similarity.

We would also like the queries to have little term overlap, so that the corresponding result set is orthogonal. Thus, we would like the term overlap to be below 0.333. Figure 2 (right) shows that until result overlap of 0.06, the running average is dominantly below 0.333.

Hence, we chose to use queries with a result overlap score in the range $(0, 0.06]$ *to find interesting orthogonal queries.*

Note that this threshold depends on the data set and should tuned for each search engine depending on the query stream.

3.3 Algorithm

To find orthogonal queries we use result overlap scores to compute query similarity between an incoming query and all queries currently in an *ad-hoc* answers cache. As just discussed, we have determined that a result overlap similarity score in the interval $(0, 0.06]$ is most beneficial for finding good orthogonal queries. Thus, we say that in practice a query A is *orthogonal* to query B if its result overlap similarity score is within this range.[1] Given a user's query, A, we construct the set of all such orthogonal queries B from the cache.

By using this cache, our algorithm is only looking at queries that have been issued within the recent past. This gives our algorithm two desirable properties. First, it can be computed online and efficiently [8] while queries are being executed, as we would not be able to compute similarity scores and present results for incoming queries if we had to run our algorithm over all previously seen queries. Second, and perhaps more importantly, our algorithm reacts to temporal changes in users' query habits. If users are currently interested in searching for `Michael Jackson died`, instead of `Michael Jackson thriller`, the orthogonal queries will reflect this fact.

4. SESSION BASED EFFECTIVENESS

In this section we aim to evaluate the effectiveness of our query recommendation algorithm with respect to others by using a large query log. Our idea is to use the sessions extracted from a query log for the evaluation purpose, where a session is a sequence of queries $q_1, q_2, ..., q_n$ submitted by a user, in a fixed amount of time t, which we will vary. That is, our training data is based in real users and not expert users.

To the best of our knowledge, this is the first time that a large scale analysis has been performed to evaluate the effectiveness of query recommendation algorithms.[2] In fact, while the use of a large scale query log has been widely proposed for training purposes, evaluation is usually done based on user judgements. We generated 7 million query recommendations for evaluation purposes, with more than 2.4 million (600 thousand for each baseline) of them for the comparison itself.

We will proceed by describing our data set, the experimental setup, the best cache policy, and the effectiveness of orthogonal query recommendation when compared with other state-of-the-art query recommendation algorithms.

4.1 Experimental Setup

We used a large search engine query log containing approximately 22 million queries from USA. We used 80% of the query log for training purposes and the rest for testing. In particular, we sessionized the testing set by using four different values for $t = 1$, 10, 20, and 30 minutes. We then consider only the *satisfied sessions with retype*, that is, sessions where the last query, q_n, is the only query that has received a click, and $n \geq 2$ (i.e. the first query of the session was not satisfactory). Furthermore, we removed sessions where q_n is not present in the training set. Table 3 reports more detailed statistics of the query log.

Number of queries	21,562,727
Number of queries with frequency = 1	4,467,362
Test corpus - number of queries	4,312,545
Training corpus - number of queries	17,250,182
Number of unique URL downloaded	34,228,300
Test Session of 1 min S-1min	99,833
Test Session of 10 min S-10min	157,747
Test Session of 20 min S-20min	170,720
Test Session of 30 min S-30min	177,183

Table 3: Statistics of the query log.

The *satisfied sessions with retype* are interesting because they simulate a real case where a user would need a recommendation. In fact, in this type of session, the user is initially unsatisfied by the URL results of q_1, and consequently, s/he tries to find an alternative query which ends in a click. We base our evaluation methodology on predicting the last query of the session. Predicting the exact last query by simply using the first query of a session is very difficult. Consider that even a single character variation between the recommended query and the last query of a session will not be considered a useful recommendation.

For evaluation purposes, we adopted the *average success at rank k* , denoted S@k, which we define to be the probability of finding at least one relevant query among the top-k recommended queries. Hence, our ground truth is very exigent. Formally, given a set of sessions $\mathcal{S}$, a session $s = (q_1, q_2, \ldots, q_n) \in \mathcal{S}$, and a set R_q of recommended queries for q_1, we define $\mathsf{S_q@k} = \begin{cases} 1, & \text{if } \{q_n\} \cap R_q \neq \emptyset \\ 0, & \text{if } \{q_n\} \cap R_q = \emptyset \end{cases}$, and with $|R_q| = k$, we define $\mathsf{S@k} = \frac{\sum_{s \in \mathcal{S}} \mathsf{S_q@k}}{|\mathcal{S}|}$.

4.2 Cache Policy

Before describing the effectiveness of orthogonal query recommendations, we discuss our static cache policy. Our goal is to keep in the cache those queries that could potentially be the final query of a *satisfied session with retype*.

While other query recommendation algorithms can benefit from the entire query log as training set for recommendation

[1] Thus, in practice, orthogonal queries are quasi-orthogonal queries in both, term overlap and result overlap.

[2] In [7, 9, 3] automated approaches are proposed, but the evaluation was based on samples of a few hundred queries.

	S-1min	S-10min	S-20min	S-30min
MCQ	**55,59**	**56,98**	**57,56**	**57,85**
MFQ	53, 08	54, 75	55, 35	55, 64
MFFQS	51, 00	52, 59	53, 12	53, 36
MRQ	34, 56	36, 27	36, 81	37, 08

Table 4: Last session query hits percentage, normalized by the maximum number of hits per session for an infinite cache. Columns are different types of sessions, rows the techniques used to fill the cache.

purposes, orthogonal query recommendation is based on a finite set of queries in the cache. In other words, a query that is recommended by the orthogonal query recommendation algorithm is, by definition, in the cache, while the other recommendation algorithms could potentially recommend a query that is chosen from the entire training set. Hence, in this sense our technique is at a disadvantage.

As we discussed in the experimental setup, we want to predict the last query of a session. Formally, given a set of queries in cache C, a session $s = (q_1, q_2, \ldots, q_n) \in \mathcal{S}$, and a set R_q of recommended queries for q_1, we define a *cache hit* if $q_n \in C$, and a *cache miss* otherwise. We selected four different features extracted from the query log: *query freshness*, *query frequency*, *query click* and *last session query likelihood*. Based on the selected features, we define:

- MCQ (*Most Clicked Query*): The most clicked queries in the training set are kept in the cache;
- MFQ (*Most Frequent Query*): The most frequent queries in the training set are kept in the cache;
- MFFQS (*Most Frequent Final Query in Session*): The query training set is sessionized, and we select the final query of each session. We keep those final queries that occur most often. Here we use sessions with $t = 30$ minutes;
- MRQ (*Most Recent Query*): The most recent queries are kept in the cache;

To avoid introducing queries into the cache that were not available during the training of the baseline algorithms, we use only the training data to determine the best cache policy. We used 80% of the query log to fill the cache to a particular size based on each policy, and we sessionized the remaining 20% for cache testing purposes.

Table 4 reports the normalized hit ratio for different cache policies and session tests by using a cache size of 80, 000 query entries. The normalization factor is obtained by considering the cache to have infinite size. The cache policy with the highest hit ratio is MCQ. Thus, the probability of having the last session of a query in the cache is higher if we adopt this policy.

Figure 4 shows the percentage of normalized query hits for different types of cache policies with varied cache sizes. The larger the cache size, the higher the hit ratio. MCQ is almost always better than any other cache policy. Sessions of 1, 10 and 20 minutes follow the same trend.

After discovering that MCQ is the cache policy with the highest hit ratio (in terms of last query of a session), we further investigate the effectiveness of the orthogonal query recommendation algorithm. We investigate the quality of the

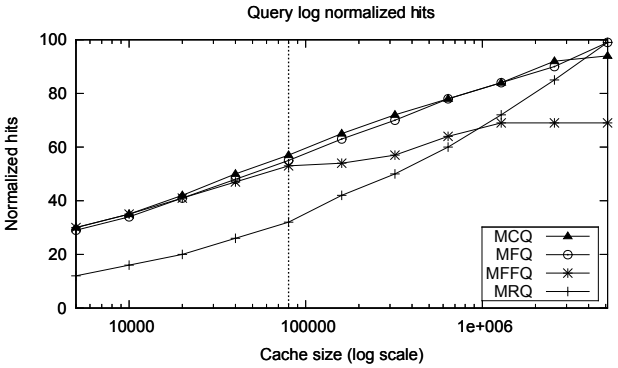

Figure 4: Percentage of S-30min query hits, normalized by the maximum number of hits reachable with a cache of infinite size.

Cache size	10, 000	20, 000	40, 000	80, 000
Sessions of 1 minute				
MCQ	2.66	**3.42**	**4.22**	**4.97**
MFQ	2.46	3.11	3.80	4.51
MFFQS	**2.78**	3.40	4.13	4.48
MRQ	0.18	0.36	0.68	1.43
Sessions of 30 minutes				
MCQ	1.95	**2.52**	**3.08**	**3.64**
MFQ	1.84	2.29	2.80	3.34
MFFQS	**2.04**	2.47	3.01	3.42
MRQ	0.14	0.27	0.51	1.05

Table 5: S@10 in percentage of orthogonal query recommendations for S-1min and S-30min. Columns indicate different cache sizes and rows the different policies used to fill the cache.

recommended orthogonal queries for different cache policies and cache sizes.

For each cache policy, the same criteria that was used to fill the cache is used to impose an order on the orthogonal queries obtained by our recommendation method.

Table 5 reports the S@10 of orthogonal query recommendations by varying the cache size and the cache policy for S-1min and S-30min. Almost always, the prediction for short session times ($t = 1$ minute) is better. This is probably due to the fact that by considering a shorter session length, the probability that the user's intent changes during the session is lower. The best cache policy is MCQ with S@10 $= 4.97$ for S-1min and S@10 $= 3.64$ for S-30min, with a cache size of 80, 000 query entries. MFFQS seems to be a good cache policy for small cache size, with a S@10 $= 2.78$ for S-1min and 2.04 for S-30min.

In conclusion, we could say that the best cache policy is MCQ, as it performs best for short term (S-1min) and long term prediction (S-30min). We note that using short term sessions (S-1min), S@10 in percentage of orthogonal query recommendations for policy MCQ appears to increase by a constant (0.75) as we double the cache size. For longer term sessions (S-30min), we see a similar trend with a smaller constant (0.56). However, as the number of queries increase and we reach the long tail, this trend will end.

Sessions	S-1min	S-10min	S-20min	S-30min
Overall				
UF-IQF	5.87	4.83	4.54	4.40
OQ	4.97	4.07	3.72	3.64
SC	3.46	2.48	2.32	2.25
SQ	1.47	1.05	0.60	0.58
CG	0.64	0.49	0.46	0.43
Unseen queries				
OQ	3.54	3.61	3.65	4.16
SC	0.78	0.80	0.79	0.79
SQ	0.74	0.72	0.72	0.71
UF-IQF	0.00	0.00	0.00	0.00
CG	0.00	0.00	0.00	0.00

Table 6: S@10 as a percentage, both for all queries and for long tail queries. In each case, the techniques are sorted in order of effectiveness.

4.3 Comparison

Having found that the best cache policy is MCQ, we now focus on comparing the effectiveness of orthogonal query recommendations w.r.t. state-of-art algorithms for query recommendation. Hereinafter, we will adopt a cache of 80,000 query entries and the MCQ policy for producing orthogonal query recommendations. Before describing our baseline algorithms, it is worth mentioning that we needed to generate approximately 600 thousand recommendations for each baseline. Hence, we selected recommendation algorithms with low computational cost.

As baselines, we used four different well-known recommendation algorithms: CG (*Cover Graph*) [3], UF-IQF (*User Frequency-Inverse Query Frequency*) [9], SC (*Short Cuts*) [7], and SQ (*Similar Queries*) [15].

Recall that these algorithms use all the queries to find the best query recommendations and our algorithm only uses the queries in the cache. Hence, our algorithm is in *disadvantage* with respect to all the baselines. However we trade-off this disadvantage with time efficiency.

Table 6 reports the S@10 as a percentage of orthogonal query recommendations compared to our baselines, for different test session lengths.[3] We report the overall results, and the results of unseen queries separately. In fact, around 21% (Table 3) of the queries in our query log are unseen queries (that is, their frequency is one).

Note that the best results are obtained with S-30min, in the case of unseen queries, and S-1min in the overall case. In the case of unseen queries, the effectiveness of OQ is higher than all of the baselines, showing its effectiveness for long tail queries. Notice that in this case, the second best algorithm is SC and not UF-IFQ, which is not able to provide recommendation for unseen queries.

The effectiveness of OQ is always higher than CG, SQ and SC. However, UF-IQF performs better than OQ when we consider the entire set of sessions. We further investigate this issue by reporting in Table 7 the percentage of the recommended queries that overlap between OQ and our baselines. That is, what percentage of the recommended queries are the same. We want to know whether the queries recommended by the classical query recommendation algorithms contain the orthogonal queries that we recommend. We restrict the test sessions to the successful sessions of each method. As we can see, only 14% of the successful recommended queries of OQ are contained in the successful recommended queries of UF-IQF. This means that the two recommendation algorithms are almost "orthogonal" in terms of recommended queries, and therefore they could be combined. The percentage of overlap with other baselines is even lower as those are not very effective; a maximum of 5% with SC, 2% with SQ, and 1% with CG.

[3]Results for S@5 and S@1 were omitted, however we did not notice any difference of order in terms of effectiveness between OQ and the baselines.

Sessions	S-1min	S-10min	S-20min	S-30min
CG	1%	1%	1%	1%
UF-IQF	14%	14%	14%	13%
SC	5%	5%	5%	4%
SQ	2%	2%	2%	2%

Table 7: Overlap of the successful orthogonal recommendations w.r.t. the successful baseline query recommendations. Columns indicate the different types of session lengths and rows are our baselines.

Judgment	useful	somewhat	not useful
Overall			
OQ	20%	25%	55%
TQG	18%	23%	59%
UF-IQF	20%	20%	60%
CG	16%	22%	62%
SC	14%	16%	70%
QFG	13%	12%	75%
SQ	6%	4%	90%
Unseen queries			
OQ	25%	8%	67%
TQG	8%	13%	79%
SC	5%	3%	92%
UF-IQF	0%	0%	100%
CG	0%	0%	100%
QFG	0%	0%	100%
SQ	0%	0%	100%

Table 8: User study results, which compare the effectiveness of OQ with the baseline techniques, sorted in order of effectiveness (useful + somewhat).

Note that OQ is not always going to be useful. Specifically, if the user's original query gives poor results, but their informational need can only be satisfied by slight variations on the original query (high result overlap), then OQ will not detect these queries. On the other hand, because OQ is the most effective method for unseen queries, in comparison with other recommendation algorithms, and as detecting such queries in the cache is trivial, it would be easy to implement OQ in conjunction with any of the other methods leading to substantial improvement in overall performance.

4.4 User Study Evaluation

In addition to the automated evaluation shown above, we also tested the effectiveness of OQ by running a user study. We experimented with the following methods: OQ, CG, UF-IQF, SQ, and SC. In addition, we introduced two

other baselines: QFG (*Query Flow Graph*) [5], TQG (*Term-Query Graph*) [6]. QFG and TQG make use of *Random Walk with Restart* for query suggestions.

The high computation time of the *Random Walk with Restart* ($\Omega(|E| + |V|)$, where $|E|$ is the number of edges of the graph and $|V|$ is the number of nodes, is why QFG and TQG have not been introduced until now.

The user study was conducted on the 50 queries from the standard TREC Web diversification task test bed.

The assessment was conducted by a group of 10 assessors. The assessors were researchers unaware of the used algorithms, and not working on related topics. We generated the top-5 recommendations per query for each technique evaluated: OQ, CG, UF-IQF, SC, SQ, QFG, and TQG.

Using a web interface, each assessor was provided with a random query from the test bed, followed by a list of recommended queries (the top-5 for each of the 7 methods) selected by the different methods. Recommendations were randomly shuffled, so that the assessors could not distinguish which method produced them. Each assessor was asked to assess each recommended query using the following scale: *useful*, *somewhat useful*, and *not useful*.[4] We gave assessors the possibility of observing the search engine results for the original query and the recommended query that was being evaluated. The user study finished when each assessor had assessed all recommendations for all 50 queries in the test bed.

Table 8 reports the results of the user study. The overall results and the results of unseen queries are reported separately. Overall, for OQ, 45% of the recommendations were judged *useful* or *somewhat* useful. This table shows that the quality of the queries recommended by OQ are higher than our baselines, and strictly higher in the *somewhat* category, which demonstrates the orthogonality of our technique. UF-IQF shows lower overall effectiveness, while only 25% of QFG recommendations were judged *useful* or *somewhat* useful. One reason for QFG's recommendation failure is the presence of high in-degree nodes. Also, QFG does not provide recommendations for unseen queries, while TQG solves this issue. In fact, for TQG, 41% of the recommendations were judged *useful* or *somewhat* useful, making it second-best in our user study. For unseen queries, 25% of OQ recommendation were judged *useful*, while for TQG only 8% were judged *useful*. In both cases, OQ was judged to be more useful than all other methods.

5. CONCLUSIONS

We presented a new query recommendation technique based on identifying orthogonal queries in an *ad-hoc* query answer cache. In contrast to previous approaches, we intentionally seek out queries that are only moderately similar to the original. Orthogonal queries aim to detect different interpretations of the user's query that go beyond the scope of the user-provided keywords.

Our approach requires no training, is computationally efficient, and can be easily integrated into any search engine with an answer cache (as answer caches are large, they can be used as a proxy cache for our algorithm).

[4]Each assessor was given the following instructions: A *useful* recommendation is a query such that if it were submitted to a search engine, it would provide URL results that were not available using the original query, and it would reflect the user's intent from the original query.

Our experimental results show that if the user is not satisfied with the original URL results of a query (i.e., he is not going to click on them), orthogonal query recommendations can satisfy the user's informational need, better than several previous approaches. Also, orthogonal query recommendation is shown to be the best technique for long tail queries, performing better than all previous techniques considered.

In the user study with the standard TREC Web diversification task test bed, orthogonal query recommendation also outperforms six known query recommendation methods, including approaches that are notably more computationally extensive. Orthogonal queries have the potential to improve other aspects of web search. Future work will investigate how they can be used in diversification of the web search results.

6. REFERENCES

[1] R. Baeza-Yates, C. Hurtado, and M. Mendoza. Query recommendation using query logs in search engines. In *Proc. EDBT'04*.

[2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval: the concepts and technology behind search*. Addison-Wesley, 2 edition, 2011.

[3] R. Baeza-Yates and A. Tiberi. Extracting semantic relations from query logs. In *Proc. KDD'07*.

[4] E. Balfe and B. Smyth. A comparative analysis of query similarity metrics for community-based web search. *LNCS*, 3620:63, 2005.

[5] P. Boldi, F. Bonchi, C. Castillo, D. Donato, and S. Vigna. Query suggestions using query-flow graphs. In *Proc. WSCD'09*.

[6] F. Bonchi, F. Silvestri, R. Perego, H. Vahabi, and R. Venturini. Efficient query recommendations in the long tail via center-piece subgraphs. In *Proc. SIGIR'12*.

[7] D. Broccolo, L. Marcon, F. Nardini, R. Perego, and F. Silvestri. Generating suggestions for queries in the long tail with an inverted index. *I.P.M.*, 2011.

[8] A. Broder. On the resemblance and containment of documents. In *Proc. SEQUENCES'97*.

[9] H. Deng, I. King, and M. R. Lyu. Entropy-biased models for query representation on the click graph. In *Proc. SIGIR'09*.

[10] B. Fonseca, P. Golgher, B. Pôssas, B. Ribeiro-Neto, and N. Ziviani. Concept-based interactive query expansion. In *Proc. CIKM'05*.

[11] M. Karimzadehgan and C. Zhai. Improving retrieval accuracy of difficult queries through generalizing negative document language models. In *Proc. CIKM'11*.

[12] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33(1):6–12, Sept. 1999.

[13] F. Silvestri. Mining query logs: Turning search usage data into knowledge. *FTIR*, 4(1-2):1–174, 2009.

[14] J. Wen, J. Nie, and H. Zhang. Clustering user queries of a search engine. In *Proc. WWW'01*.

[15] O. R. Zaïane and A. Strilets. Finding similar queries to satisfy searches based on query traces. In *Proc. OOIS'02*.