# Query Suggestion Using Anchor Text

Van Dang
CIIR
Department of Computer Science
University of Massachusetts
Amherst, Massachusetts
vdang@cs.umass.edu

W. Bruce Croft
CIIR
Department of Computer Science
University of Massachusetts
Amherst, Massachusetts
croft@cs.umass.edu

Andrew McGregor
CIIR
Department of Computer Science
University of Massachusetts
Amherst, Massachusetts
mcgregor@cs.umass.edu

## ABSTRACT

Many query suggestion techniques have been proposed to better capture user intent and to improve search effectiveness. However, most of these methods make use of query logs which are not readily available to the research community. Anchor text, on the other hand, is widely available and has proven useful for many tasks. In this paper, we investigate the problem of query suggestion via random walk and demonstrate that anchor text can be an excellent substitute for a query log. In particular, our results suggest that anchor text is as effective as a real log for this technique and it even outperforms the real log in some cases. In addition, we propose a simple but efficient implementation of the random walk procedure under the MapReduce framework. Each iteration of the walk can be done in a matter of minutes on graphs with million of vertices.

## Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Query Formulation

## General Terms

Algorithms, Measurement, Performance, Experimentation.

## Keywords

Query suggestion, query log, anchor text, random walk, MapReduce.

## 1. INTRODUCTION

Many different query formulations are possible for a given information need. Some of those formulations will be effective, others will not. Formulating the correct query for web search can be a difficult task, given that users may not be aware of the right vocabulary to use. As a result, query suggestion has become an important part of web search engines.

Query reformulation or query rewriting techniques [10, 20, 23] manipulate the original query by replacing words or adding new words in order to create a more effective query. In contrast, query suggestion or query recommendation techniques [2, 24, 1, 16] use the whole original query to find closely related queries that have previously been submitted to the search engine. While each class of technique has advantages, we focus on query suggestion techniques in this paper.

Most of the proposed query suggestion techniques make use of query logs as the source of past observed queries. While some methods find suggestions for a query based on the content similarity between their clicked documents [24, 1], in this paper, we investigate a "content-ignorant" approach because it is more practical on large datasets. In particular, we focus on random walk-based methods such as that proposed by Mei and Church [16]. In general, these methods build a bi-partite click-graph from a query log and use queries connected via a random walk as suggestions for each another. The expected number of steps in getting from one query to another is the "score" for the suggestion. The key advantage of this type of method is its ability to find suggested queries that do not have any clicked documents in common with the initial query.

A major problem with this technique, and others, is that query logs are not readily available to the research community. This makes it difficult to compare the relative effectiveness of techniques and apply them in domains for which there is no query log. To address this lack of availability, we propose to use anchor text to simulate parts of a log since they have very similar structure. Anchor text is often a concise description of the page it points to and can be thought of as a "query" to retrieve that page. The similarity between anchor text and real queries has also been observed by other researchers. Eiron and McCurley [8] showed that anchor text and real queries are similar regarding term distribution and length. Nallapati et al. [19] used anchor text as queries to train retrieval models. More recently, Dang and Croft [6] have shown that anchor text is as effective as a real query log for the task of query reformulation. In this paper, we create an anchor log from anchor text extracted from the TREC ClueWeb collection. Suggestions obtained with this anchor log and a real log (the MSN log [18]) via random walks are compared in terms of retrieval performance on TREC collections. Our results show that the anchor log is at least as effective as the real log, and can significantly outperform it in one query set.

Random walk is a standard graph algorithm that has many applications in the area of information retrieval as pointed out by Craswell and Szummer [4]. In addition to query suggestion, it has also been applied to document ranking [4], label propagation [22, 11, 13] and click-through data smoothing [9]. Existing work, however, has either used a relatively small log or performed only short walks. When logs get larger and longer walks become necessary, the efficient implementation of the random walk is an important issue that does not appear to have been sufficiently addressed in previous research. In this paper, we propose a simple implementation for the random walk procedure using the MapReduce framework.

In the next section, we describe the process of building the anchor log from a web collection and compare it with the MSN log. Section 3 presents the random walk model for query suggestion and its implementation under the MapReduce framework. Section 4 contains the experimental results as well as our discussion of them. Section 5 describes related work in the area and finally, Section 6 will conclude.

## 2. ANCHOR LOG CONSTRUCTION

A query log contains a broad range of information gathered from searchers. The most important part for query suggestion is the queries issued by users and the documents they clicked on for those queries. Session information which captures users' behaviour either within a window of time or of other segmentation criteria is also useful for a variety of tasks. However, the technique that we investigate in this paper does not use session information.

Even though search logs have proven important in many ways, they generally are not available to the research community due to privacy concerns. On the other hand, anchor text, which is widely available, has very similar structure to a search log and thus potentially can be a substitute. In fact, Dang and Croft [6] have shown that a log simulated from anchor text is as good as a real log for query reformulation.

Web pages are connected to one another via hyper-links, each of which is associated with some anchor text. A link is called *internal* if two connected pages are from the same domain and *external* if they come from different domains. Since most of the internal links are for navigation purposes, their associated anchor text is not very helpful. Typical examples of such anchor text are "home" and "index". As a result, we only consider *external* links in our construction process.

The web collection for anchor text mining that we use is the English portion of the ClueWeb-09 catetory A [1]. It contains 500 million pages in English that were crawled from the web during early 2009. We extracted all pairs of anchor text and associated urls from the web pages in this collection.

In order to reduce noise, we discarded anchors that contain non-English words and those that contain navigation-triggered words such as "click", "download" and "subscribe". We also removed anchors that contain only numbers and stop words. We put the resulting <anchor text, url> pairs together to build a simulated query log, which will be refered to as the anchor log in the rest of the paper. In the anchor log, anchor text replaces the queries, and hyperlinked pages represent clicked documents. Our anchor log is similar to that of Dang and Croft [6] but on a larger scale.

[1]http://boston.lti.cs.cmu.edu/Data/clueweb09/

**Table 1: Statistics of MSN Log and Anchor**

| | MSN Log | Anchor Log |
|---|---|---|
| # Total Queries | 12,250,998 | 5,484,766,989 |
| # Unique Queries | 3,544,809 | 323,640,671 |
| Average Query Length | 2.51 | 2.74 |

We use the MSN log [18] as the real log counterpart for comparison. The MSN log is a sample of queries submitted to a commercial search engine over a month period. The same content filter was applied to this log. Table 1 shows the statistics of these two logs.

One might argue that the real log is at a disadvantage in this comparison considering its size. This, in fact, reinforces our point about the potential of anchor text: it is freely available virtually in any amount. Since the goal of this study is to explore the possibility of using anchor text as a substitute for query logs, the difference in size should not matter.

## 3. RANDOM WALK ON A CLICK GRAPH

Random walk is a general procedure that can be applied on graphs. It has two main variations: the *forward* walk model computes the probability of getting from point $A$ to point $B$ in $t$ steps while the *backward* walk model computes the probability of reaching $B$ from $A$. Therefore, the *forward* model can be explained as *predictive* and the *backward* model as *diagnostic*. The model applied in this paper is the *forward* walk.

### 3.1 The Model

A bipartite graph is a graph where vertices can be organized into two sets such that there are no edges connecting vertices in the same set. The click graph is such a bipartite graph with queries on one side and documents on the other. An edge connecting a query and a document indicates that we have observed clicks for that document-query pair and its weight is the number of clicks. In the case of the anchor log, the edges in the graph represent <anchor text, url> pairs.

With the click graph in mind, the random walk procedure can be explained as follows. Starting at a query $q_i$, a user can "walk" to connected documents, each with some probability. At each document, the user then "walks" to other connected queries and the process repeats. The fact that one can "walk" from $q_i$ to $q_j$ indicates that $q_j$ is relevant to $q_i$ and thus, $q_j$ can be potentially suggested to users who issue $q_i$.

The process above forces the user to move to a different vertex in every step. Thus longer walks can get further away from the starting point. In the context of our query log click graph, it is unlikely that queries that are too far from the initial query make good suggestions. Therefore, we allow self transition in the random walk process. In other words, at every step, the user can decide whether to move to another

place or stay in the current place. Putting more emphasis (e.g. high probability) on self transition promotes queries closer to the original.

## 3.2 Computation

Let $G = \langle Q, D, E \rangle$ be our bipartite click graph where $Q$ is the set of queries, $D$ is the set of documents and $E$ is the set of edges. Let $e_{ij} \in E$ be some edge connecting the query $i \in Q$ and the document $j \in D$ and $c_{ij}$ be the number of clicks (or the number of links for the anchor log) we have observed for the pair $\langle i, j \rangle$. The probability of walking from $i$ to $j$ is given by:

$$P_{ij} = \frac{c_{ij}}{\sum_{k \in N_i} c_{ik}}$$

where $N_i$ is the set of neighbors of $i$. Due to normalization, note that while $c_{ij}$ is symmetric, $P_{ij}$ is not.

Let $s$ be the self-transition probability. Since we allow a walk to stay at its current node with probability $s$, the probability of the walk from $i$ to $j$ becomes:

$$P_{ij} = \begin{cases} (1-s)\frac{c_{ij}}{\sum_{k \in N_i} c_{ik}} & \text{if } i \neq j \\ s & \text{if } i = j \end{cases}$$

## 3.3 Basic Implementation

The most straightforward method to implement this random walk procedure is through in-memory matrix multiplication. We can represent the one-step transition probabilities as a matrix $T$. Each starting query can be encoded as a vector $q_i$ recording the probabilities of transiting to all nodes in the graph. Let $q_i^0$ be the initial distribution of the query $i$. This distribution after a $t$-step walk is:

$$q_i^{(t)} = q_i^{(t-1)} \times T$$

This implementation might work very well on small datasets. When graphs get larger, however, the matrix will not fit in memory and efficiency as well as scalability of this implementation becomes a problem which has not yet been discussed. Gao et al. [9] conducted random walks on a relatively large graph but they performed only a 2-step walk. Mei and Church applied random walk on a graph with hundreds of millions of vertices, yet how the graph was filtered was not discussed. Another factor that needs to be addressed is the growth rate of reachable vertices. A 101-step walk conducted by Craswell and Szummer [4] retrieves only about $1,000$ vertices whereas a 5-step walk on our anchor graph can reach 5 million other nodes.

## 3.4 MapReduce Implementation

MapReduce is a framework proposed for distributed processing [7]. It includes Mappers that split the input data into different streams and "map" each of them to a worker node. The processing is thus carried out simultaneously at different nodes to produce intermediate results which are then aggregated by Reducers to produce the final output. A detailed introduction of MapReduce is out of the scope of this paper. Instead, we only provide such a brief description and focus on how to implement the random walk procedure under this framework and what we can benefit from it.

The key requirement for a procedure to be beneficial under the MapReduce framework is that its processing must be able to be applied locally on different parts of the input data, thus allowing multiple processes to be done at the same time. Fortunately, the random walk procedure is an excellent fit for this requirement.

Let $q_A^{(t-1)}$ be the distribution over vertices at which a walk starting at vertex $A$ after $(t-1)$ steps can be and $T$ be the matrix recording one-step transition probabilities. Assuming that at time step $(t-1)$, only vertices $B$, $C$ and $D$ are reachable from $A$ as given in Fig. 1. At time step $t$, $A$ can now reach $E$ via all $B$, $C$ and $D$ and the probability $P_{AE}$ is given by:

$$P_{AE} = P_{AB} \times P_{BE} + P_{AC} \times P_{CE} + P_{AD} \times P_{DE} \qquad (1)$$

Since the three product terms in the right hand side of Eq. 1 are independent of each other, their computation can be done in parallel by different worker nodes. This leads to our implementation as shown in Fig. 2.

Two mappers in the first layer will read in pairs of vertices: one set from $q_A^{(t-1)}$ and the other from $T$. Note that while pairs in $q_A^{(t-1)}$ indicate path from the start vertex to the target vertex, those in $T$ indicate edges connecting the two vertices. For the ease of presentation, we refer to both types of pairs as "edges". The mappers in the first layer distribute these edges to those in the second layer. It is worth mentioning that we need to configure $Mapper^{(1-1)}$ and $Mapper^{(1-2)}$ such that edges from the two sets that are to be connected (such as $A \to B$ and $B \to E$) will be distributed to the same mapper. Each mapper in the second layer simply does a join operation to connect a pair of incoming edges to emit a new one. Finally, the reducer will aggregate all edges that share both source and target vertices. Pseudo code for this procedure is provided in Table 2. Note that we ignore probability normalization in the Reducer class for simplicity.

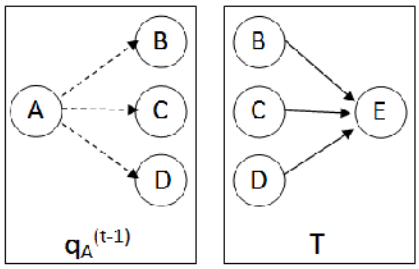



**Figure 1: An illustrating example for implementing random walks in the MapReduce framework. Solid arrows represent edges and dashed arrows represent paths.**

## 3.5 Controlling the Exponential Growth

Due to the nature of the walk process as described in section 3.1, the number of reachable nodes can increase exponentially with the length of the walk, which certainly raises the efficiency issue when long walks are necessary. The issue is especially bad with dense graphs such as our anchor graph, as will be shown in section 4.4.

As we mentioned above, the further a query is from the original, the less likely will it make a good suggestion. Therefore, among the exponential number of queries discovered as we

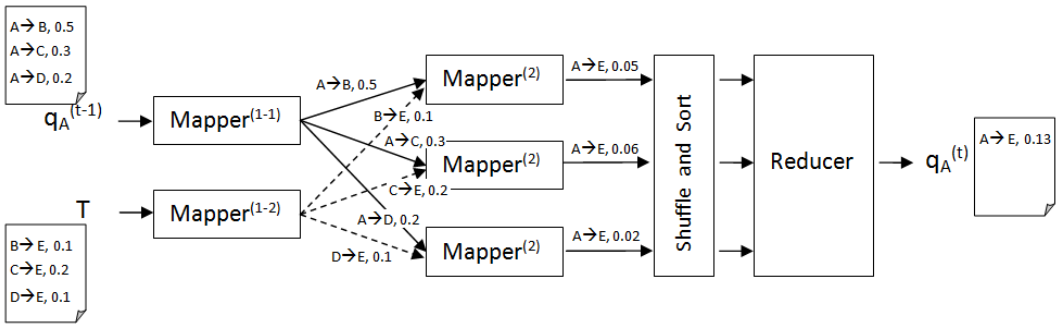



**Figure 2: One iteration of random walk in the MapReduce framework. Pair of edges to be connected are distributed to the same mapper which joins them based on the common vertex to emit a new edge. The reducer aggregates intermediate instances of "A →E" to generate the complete edge.**

**Table 2: Pseudo code for the random walk procedure in MapReduce. Note that we ignore the probability normalization in the Reducer class for simplicity.**

| Mapper$^{(1-1)}$ |
|---|
| Input: $q_A^{(t-1)}$<br>For each edge $e$ in $q_A^{(t-1)}$<br>  Emit $e$<br>End For |
| Mapper$^{(1-2)}$ |
| Input: $T$, self-transition probability $s$<br>For each group of edges $G$ in $T$ that shares the source vertex<br>  For each edge $e$ in $G$<br>  $e.prob \leftarrow e.prob \times (1-s)$<br>  Emit $e$<br>  End For<br>  Emit new Edge($G.sharedVertex$, $G.sharedVertex$, $s$)<br>End For |
| Mapper$^{(2)}$ |
| Input: stream $s_1$ from $q_A^{(t-1)}$, stream $s_2$ from $T$<br>For each pair $e_1 \in s_1$ and $e_2 \in s_2$<br>such that $e_1.target = e_2.source$<br>  Emit new Edge($e_1.source$, $e_2.target$, $e_1.prob \times e_2.prob$)<br>End For |
| Reducer |
| Input: List of edges $E$ sorted by both *source* and *target*<br>$G \leftarrow$ edges that share both source and target vertex<br>$e \leftarrow$ any $e' \in G$<br>$e.prob = \sum_{e' \in G} e'.prob$ |

extend the walk, most of them will unlikely be helpful. As a result, it is not necessary to maintain all of them. In this paper, we propose to maintain only top-$K$ new vertices at each step from which we will allow further walk. We can either set $K$ to a constant or lower its value as we walk further from the starting point. For simplicity, we choose the former approach.

Since the set of reachable nodes at the end of each step includes both old vertices (because we allow self-transition) and new vertices. Simply keeping only top-$K$ might prevent us from getting any new vertices since the top-$K$ might be dominated by old vertices, especially when we use a reasonably large self-transition probability to promote suggestions closer to the original query. As a result, we record only the top-$K$ among newly discovered vertices and keep all of the old ones.

**Table 3: Statistics of the click graphs built from the MSN Log and our Anchor Log**

| | #queries | #urls | #edges |
|---|---|---|---|
| MSN Log | 826,639 | 541,352 | 1,609,827 |
| Anchor Log | 8,215,751 | 9,885,766 | 29,811,501 |

# 4. EXPERIMENTS

In this section, we evaluate the efficiency of our proposed implementation of the random walk technique as well as compare the effectiveness of our anchor log and the MSN log for query suggestion.

## 4.1 Data Preparation

We construct a bi-partite graph for each of the logs. Since <query, url> pairs that have low frequency might not be reliable, we filter out all pairs with frequency of 1. In an attempt to reduce noise, we follow Craswell and Szummer [4] by doing a two-stage pruning of the graph. We first remove all urls that are connected to only one query and then remove all queries that are connected to only one url. The statistics of our pruned graphs is shown in Table. 3.

Table. 3 suggests that the anchor graph is denser than the MSN graph. The $\#queries/url$ and $\#url/query$ ratio for the MSN graph are 2.97 and 1.95 respectively while these ratios for the anchor graph are 3.02 and 3.62.

## 4.2 Parameter Settings

In our preliminary experiments, we tried different values for the self-transition probability. This is by no mean an exhaustive parameter tuning. We observed the best performance with $s = 0.4$ for the MSN log and 0.01 for the anchor log. Thus, we will use these values in all of our experiments.

## 4.3 Evaluation Method

It is always difficult to evaluate query suggestion techniques since the definition of "quality" of suggestions is unclear. Beeferman and Berger [2] evaluate suggestions based on their actual click-through rate. Jones et al. [10] evaluate suggestions by manually checking how similar they are to the original queries. In this paper, we choose to evaluate suggestions using their retrieval performance instead.

To simulate the context of web search, we do retrieval on the ClueWeb09 category A dataset. It contains roughly 500 million web pages crawled during January and February 2009. We use two different query sets, one from the TREC Web Track 2009 (*WT-09*) and the other from TREC Web Track 2010 (*WT-10*). The former set contains 50 queries and the latter has 36. It is worth mentioning that the relevance judgments are incomplete due to the size of this collection.

The core evaluation task in this study is that given two different lists of suggestions for the same query, how should we determine if they are good and which one is better? We could of course judge a list by its top suggestion: it is good if that suggestion performs better than the original query with respect to some retrieval metric. However, to the best of our knowledge, there is no known techniques that can consistently provide a single suggestion that is better than the initial query. Since our goal in this study is to compare the anchor text and query logs rather than trying to optimize the query suggestion method, we choose to use a more relaxed criteria.

We do our evaluation as following. We use the original query and top-$m$ suggestions in each list to do retrieval and record their $MAP$ and $NDCG$@10. Then we manually choose the best suggestion among each list as its representative, and thus the $MAP$ or $NDCG$@10 of a list will be that of its best suggestion. Now we can compare one list to the other as well as to the original query. We vary $m$ from 1 to 10 in our experiments. This can be explained in the context of a user using a search engine: the search engine suggests a list of alternatives to the user and the user will likely be able to select good suggestions. Hence, the list can be considered "good" if it contains a "good' suggestion.

It is worth noting that the goal of suggesting alternative queries is not always to improve the original query, but sometimes it is to suggest queries on related topics. In this paper, however, we assume that we are trying to improve the original query and leave related query suggestion to future work.

We used Lemur/Indri[2] as the retrieval software and language modeling [5] was used for retrieval. When any suggested query is used to do retrieval, it is always combined with the initial query via the Indri's *#combine* operator with equal weight. We use TupleFlow [21] as an implementation of MapReduce. A two-tailed t-test is used with $p\text{-}value < 0.05$ to perform significance tests.

## 4.4 Efficiency

The main efficiency problem for the implementation of random walks is the growth rate of reachable nodes over time.

[2]http://www.lemurproject.org

**Table 4: The number of reachable vertices at different walk's lengths. While this growth is managable with the MSN log, this is not the case with the anchor log. "Anc-" stands for the anchor log and "MSN-" represents the MSN log.**

| L. | Anc-Full | Anc-Top300 | MSN-Full | MSN-Top300 |
|---|---|---|---|---|
| 2 | 1,655 | 819 | 56 | 50 |
| 3 | 134,277 | 1,119 | 178 | 117 |
| 4 | 457,435 | 1,419 | 695 | 219 |
| 5 | 3,548,565 | 1,719 | 1,571 | 332 |
| 6 | 6,823,483 | 2,019 | 4,372 | 486 |

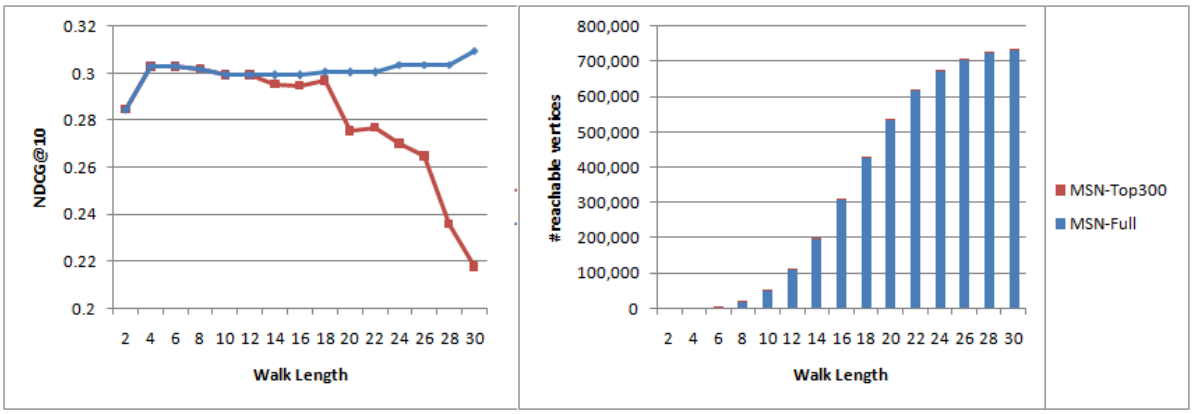
Intuitively, the denser the graph, the higher is this growth rate. Table 4 shows the average number of reachable nodes for one initial query when using the standard procedure (presented as "-Full") and our *top-K* approach (marked as "-Top300") in which we set $K = 300$.

While the growth on the MSN log is manageble due to its small size, this is not the case with the anchor log. After only 6 steps, the walk covers more than one third of the graph. Keeping only *top-K* new vertices at each step certainly reduces this growth. However, since random walk works by propagating probabilities among all connections, this filtering will direct all the probability mass to those *top-K* vertices. As a result, it will affect the walk's behaviour.

To verify if simply maintaining only the top-$K$ new vertices at each step is reasonable, we compare it with the standard procedure on the MSN log with a 30-step walk. The self-transition probability is set to $t = 0.4$. For simplicity, we only present the results observed with *WT-09* since the results with *WT-10* are very similar. Quality of suggestions are evaluated as described in section 4.3 with $m = 10$.

Fig. 3 shows the average number of reachable vertices per initial query and the quality of suggestions at each iteration of the two procedures. We can see that while the *Top300* approach only maintains a very small set of reachable vertices, its suggestions are at least as good as those generated by the standard procedure that has to maintain many more reachable vertices, and thus is more computationally expensive as can be seen in Fig. 4 in terms of running time. In addition, it is interesting to notice whereas longer walks with the standard procedure result in lower quality suggestions, this is not the case with the *Top300* approach. This helps validate our intuition to some degree that when a node $p$ is connected with too many nodes, not all of these connections are helpful. In fact, propagating the probability mass of $p$ along all these connections might diffuse the walk away from good suggestions. This is especially true when $p$ is further away from the starting query, which is why the performance drop of the standard procedure starts from the $20th$ iteration but not early in the process.

For the rest of the paper, we will apply this *Top-300* approach to the procedure on the anchor log since running the standard procedure is not practical considering we will experiment with long walks. With the MSN log however, we will use the standard walk procedure. The reason is that, even though the *Top-300* approach is more stable with the




(a) NDCG@10 (b) Number of reachable vertices per query

**Figure 3: Comparison of the standard random walk procedure and our *Top-300* approach. Fig. (a) shows the *NDCG@5* for suggestions provided by the two procedures (for 32 queries in *WT-09*) and Fig. (b) shows the number of reachable vertices per starting query at each iteration that needs to be maintained by each procedure. While our approach maintains a much smaller set of reachable vertices than the standard procedure, the suggestions that it generates are at least at good.**

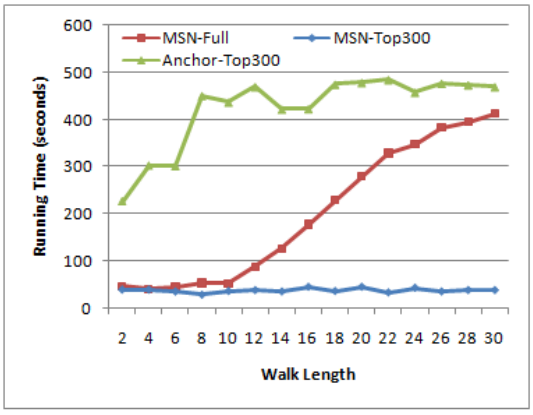



**Figure 4: The running time of each step of the random walk on two logs.**

length of walk length, the best results we achieved with the MSN log in most cases are observed to be from the standard procedure (though the differences are not significant). Given our evaluation criteria, using the standard procedure will give the MSN log some advantages.

The running time [3] of these three processes is given in Fig. 4. Each iteration of the algorithm (starting at all queries at the same time) takes an average of 38 seconds on the MSN graph and 423 seconds on the anchor graph using *Top300*. We note that we can reduce the running time by using a more efficient data structure to store the graph on disk. In this work, our graphs are encoded as lists of edges and they are stored as simple gzipped text files. Therefore, reading the graph from disk is responsible for a large portion of the running time for the anchor log graph.

[3]Our hardware cluster includes 60 compute nodes with 8 cores (Xeon 5355 2.66 GHz), 16GB of ram. However, we normally used only 40 mappers in our experiments.

## 4.5 Query Suggestion Evaluation

In this section, we will evaluate the effectiveness of the anchor log and the MSN log for query suggestion. Experiments proceed as follows. For each query $q$ in our query set that appears in the log, we use it as the starting point from which we conduct a random walk of length $t$. We set $t$ to 30 in all of our experiments since we have observed no significant improvement with larger $t$. At the end of the walk, the top-$m$ query nodes that are reachable from $q$ are considered suggestions generated by this log. This list is compared to the original query as well as other lists using the evaluation method described in section 4.3 above.

### 4.5.1 *The Anchor Log*

We were able to obtain suggestions to 40 out of 50 queries from the web track 2009's query set and 31 out of 36 from the 2010's set from our anchor log. We set $m = 10$ in this experiment. Table 5 shows the performance comparison between the suggestions generated with this log and the initial queries.

These results across two query sets consistently show that suggestions provided by this log significantly outperform the original queries, regardless of the walk length. Most of the differences are statistically significant at $p\text{-}value < 0.05$. This suggests that the log is indeed very effective in suggesting queries with high retrieval performance. Some examples of these suggestions are provided in Table 6.

It is interesting to see that the best performing suggestions were generated very early in the walk process: at $t = 2$ on *WT-09* and $t = 4$ on *WT-10*.

### 4.5.2 *The MSN Log*

The MSN log can only provide suggestions for 34 queries in *WT-09* and to 26 queries in *WT-10*. Its performance is shown in Table 7. Similar to the results with the anchor log, the suggestions generated with the MSN log are also

**Table 6: Examples of suggestions generated with the anchor log. The first column shows the original query of which *NDCG@10* is given in the second column. The suggestions to this query together with its *NDCG@10* is provided in the third and fourth column respectively.**

| Orig. Query | NDCG@10 | New Query | NDCG@10 |
|---|---|---|---|
| neil young | 0.4254 | neil young living with war | 1.0 |
| bellevue | 0.103 | www ci bellevue wa us | 0.6823 |
| tornadoes | 0.468 | how tornadoes work | 0.7382 |
| ocd | 0.041 | obsessive compulsive disorder | 0.2107 |
| kcs | 0.0 | kansas city southern | 0.4026 |
| joints | 0.0 | joint pain | 0.0214 |
| air travel information | 0.0 | permitted and prohibited items | 0.0245 |
| disneyland hotel | 0.0231 | discount hotel anaheim california | 0.1784 |
| iron | 0.0 | wikipedia iron | 0.4362 |
| the music man | 0.0 | professor harold hill | 0.3577 |

**Table 5: Effectiveness of the anchor log for query suggestions. The 3rd row provides the retrieval scores of the original queries in *WT-09* and *WT-10*. Subsequent rows present results for suggested queries obtained from this log at different walk-length. "rw-n" indicates a "n"-step walk. The suggested queries are better than the original in both metrics. $^*$ indicates statistically significant difference to the original queries at *p-value*$< 0.05$.**

| | *WT-09* | | *WT-10* | |
|---|---|---|---|---|
| | NDCG@10 | MAP | NDCG@10 | MAP |
| Orig. Q. | 0.233 | 0.0696 | 0.1675 | 0.1103 |
| rw-02 | 0.3505$^*$ | 0.1112$^*$ | 0.2907$^*$ | 0.1464$^*$ |
| rw-04 | 0.3434$^*$ | 0.1061$^*$ | 0.309$^*$ | 0.1555$^*$ |
| rw-06 | 0.3294$^*$ | 0.1031$^*$ | 0.2878$^*$ | 0.1472$^*$ |
| rw-08 | 0.3319$^*$ | 0.1031$^*$ | 0.2878$^*$ | 0.147$^*$ |
| rw-10 | 0.3256$^*$ | 0.0993$^*$ | 0.2781$^*$ | 0.1463$^*$ |
| rw-12 | 0.3214$^*$ | 0.0993$^*$ | 0.2685$^*$ | 0.1428$^*$ |
| rw-14 | 0.3198$^*$ | 0.0979$^*$ | 0.2785$^*$ | 0.1466$^*$ |
| rw-16 | 0.3198$^*$ | 0.0965$^*$ | 0.273$^*$ | 0.1463$^*$ |
| rw-18 | 0.3204$^*$ | 0.0973$^*$ | 0.2761$^*$ | 0.1486$^*$ |
| rw-20 | 0.3197$^*$ | 0.0973$^*$ | 0.2761$^*$ | 0.1486$^*$ |
| rw-22 | 0.3209$^*$ | 0.0967$^*$ | 0.2761$^*$ | 0.1486$^*$ |
| rw-24 | 0.2992$^*$ | 0.0903 | 0.2761$^*$ | 0.1486$^*$ |
| rw-26 | 0.2986$^*$ | 0.0902 | 0.2798$^*$ | 0.1497$^*$ |
| rw-28 | 0.3003$^*$ | 0.088 | 0.2798$^*$ | 0.1497$^*$ |
| rw-30 | 0.2998$^*$ | 0.088 | 0.2804$^*$ | 0.1559$^*$ |

**Table 7: Effectiveness of the MSN log for query suggestions. The 3rd row provides the retrieval scores of the original queries in *WT-09* and *WT-10*. Subsequent rows present results for suggested queries obtained from this log at different walk-length. "rw-n" indicates a "n"-step walk. The suggested queries are better than the original in both metrics. $^*$ indicates statistically significant difference to the original queries at *p-value*$< 0.05$.**

| | *WT-09* | | *WT-10* | |
|---|---|---|---|---|
| | NDCG@10 | MAP | NDCG@10 | MAP |
| Orig. Q. | 0.2121 | 0.0661 | 0.1839 | 0.1396 |
| rw-02 | 0.2845$^*$ | 0.0783 | 0.3126$^*$ | 0.1781$^*$ |
| rw-04 | 0.3027$^*$ | 0.0828 | 0.3161$^*$ | 0.1774$^*$ |
| rw-06 | 0.3027$^*$ | 0.0828 | 0.3161$^*$ | 0.1773$^*$ |
| rw-08 | 0.3017$^*$ | 0.0826 | 0.3161$^*$ | 0.1773$^*$ |
| rw-10 | 0.2992$^*$ | 0.0828 | 0.3161$^*$ | 0.1773$^*$ |
| rw-12 | 0.2992$^*$ | 0.0839 | 0.3161$^*$ | 0.1764$^*$ |
| rw-14 | 0.2953$^*$ | 0.0829 | 0.3161$^*$ | 0.1764$^*$ |
| rw-16 | 0.2945$^*$ | 0.0826 | 0.3197$^*$ | 0.1771$^*$ |
| rw-18 | 0.2968$^*$ | 0.0828 | 0.3205$^*$ | 0.1773$^*$ |
| rw-20 | 0.2754$^*$ | 0.081 | 0.3205$^*$ | 0.1773$^*$ |
| rw-22 | 0.277$^*$ | 0.0807 | 0.3169$^*$ | 0.1766$^*$ |
| rw-24 | 0.2699$^*$ | 0.0809 | 0.3144$^*$ | 0.1759$^*$ |
| rw-26 | 0.2646$^*$ | 0.0781 | 0.2642$^*$ | 0.1609 |
| rw-28 | 0.2359 | 0.0723 | 0.224 | 0.1242 |
| rw-30 | 0.2176 | 0.0711 | 0.2247 | 0.1255 |

significantly better than then original queries. Examples of suggestions provided by this log are given in Table 8.

### 4.5.3 Log Comparison

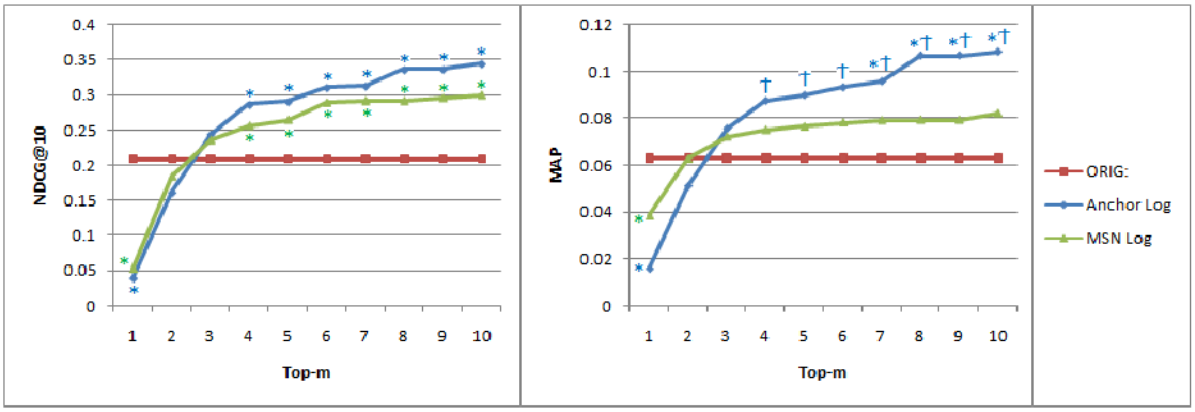
In this experiment, we use only the subset of queries of which suggestions can be generated with both logs. This results in a set of 32 queries for *WT-09* and 25 queries for *WT-10*. In addition, we set the walk-length parameter $t = 2$ and $t = 4$ for the anchor log on *WT-09* and *WT-10* respectively; $t = 4$ and $t = 18$ for the MSN log on *WT-09* and *WT-10* respectively since these settings provide the best results in previous experiments. We vary $m$ from 1 to 10 to see where the effective suggestions are in the each list.

Fig. 5 and Fig. 6 show the performance comparison between the two logs on *WT-09* and *WT-10* respectively. Suggestions generated by both logs again outperform the original queries both in $NDCG@10$ and $MAP$ starting at $m = 3$ on both query sets. This means effective suggestions are among the top-3 of each list. Statistically significant improvements occur from $m = 4$.

While both logs perform comparably on *WT-10*, the anchor log even achieves much higher $NDCG$ and significantly higher $MAP$ than the MSN log. This confirms out intuition that anchor text can be an excellent substitute for a query logs for query suggestion. This result is also consistent with that of Dang and Croft [6] which shows that anchor text is as effective as a query log for query reformulation.

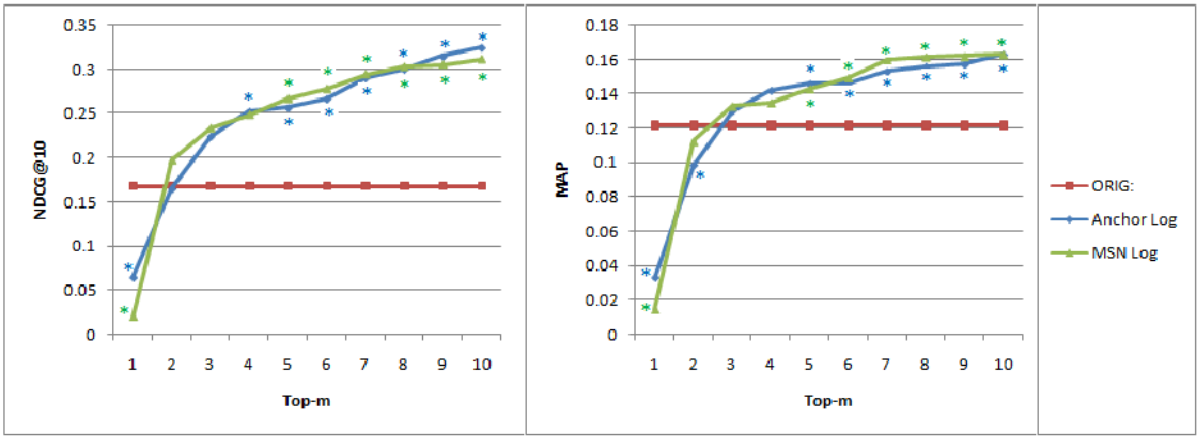
Table 8: Examples of suggestions generated with the MSN log. The first column shows the original query of which *NDCG@10* is given in the second column. The suggestions to this query together with its *NDCG@10* is provided in the third and fourth column respectively.

| Orig. Query | NDCG@10 | New Query | NDCG@10 |
|---|---|---|---|
| moths | 0.1714 | scientific name for a moths | 0.389 |
| iron | 0.0 | iron and its oxygen carrying capacity in the human body | 0.7315 |
| getting organized | 0.2341 | organizedhome.com | 0.4603 |
| arizona game and fish | 0.164 | arizona fishing | 0.1712 |
| kcs | 0.0 | kansas city southern | 0.4026 |
| afghanistan | 0.0367 | afghanistan natural resources | 0.0991 |
| used car parts | 0.1197 | used auto parts | 0.3083 |
| volvo | 0.3317 | volvo cars | 0.63 |
| flushing | 0.0 | flushing city council | 0.6333 |
| map | 0.0 | www.mapquest.com | 0.0792 |




(a) NDCG@10 (b) MAP

Figure 5: Performance comparison among the original queries in the *WT-09* set and their suggestions generated with both logs. Fig. (a) presents the differences in *NDCG@10* and Fig. (b) presents those in *MAP*. Both logs are able to generate suggestions with much higher retrieval effectives than the original queries and the anchor log even outperforms the MSN log regarding this task. $^*$ and $^\dagger$ indicate significant difference to the original queries and suggestions generated by the MSN log respectively at $p\text{-}value < 0.05$.




(a) NDCG@10 (b) MAP

Figure 6: Performance comparison among the original queries in the *WT-10* set and their suggestions generated with both logs. Fig. (a) presents the differences in *NDCG@10* and Fig. (b) presents those in *MAP*. Both logs are able to generate suggestions with higher retrieval effectives than the original queries. The two logs perform comparably on this query set. $^*$ indicate significant difference to the original queries at $p\text{-}value < 0.05$.

**Table 9: Performance comparison separated by "Help" and "Fail". Rows marked with "Help" and "Fail' represent the subset of queries for which the log can provide effective suggestions and those that it fails to provide any effective suggestion respectively. The 2nd and 3rd column provide the number of queries in each set and their *NDCG@10*. The 4th and 5th column present *NDCG@10* of suggestions as well as the relative improvement over the original queries.**

| | | #q | Orig.Q | Suggestion | %Increase |
|---|---|---|---|---|---|
| WT-09 | Anchor Log | | | | |
| | Help | 24 | 0.1859 | 0.28 | +50.62% |
| | Fail | 2 | 0.7164 | 0.0403 | -94.38% |
| | MSN Log | | | | |
| | Help | 23 | 0.1959 | 0.2454 | +25.24% |
| | Fail | 5 | 0.433 | 0.0545 | -87.42% |
| WT-10 | Anchor Log | | | | |
| | Help | 19 | 0.1888 | 0.3031 | +60.55% |
| | Fail | 1 | 0.1432 | 0.0032 | -97.76% |
| | MSN Log | | | | |
| | Help | 20 | 0.1863 | 0.2922 | +56.85% |
| | Fail | 0 | 0.0 | 0.0 | +0% |

To understand the difference in performance of the two logs, we divide our query set into two: one subset contains queries for which the log can generate suggestions with higher *NDCG@10* than the initial queries and the other subset contains queries for which the log fails to provide any such suggestions. We then examine the performance of both logs with respect to each group. Results are presented in Table 9.

With the *WT-09* query set, though both logs are able to provide effective suggestions for roughly the same number of queries, the suggestions obtained from the anchor log are indeed more effective since they show an improvement of 50.62% over the original queries whereas those from the MSN log provides only 25.24%. In addition, the MSN log also fails with more queries than the anchor log. These results explain why the anchor log achieves significantly better performance than the MSN log with *WT-09*. With *WT-10*, however, both logs help and fail about the same number of queries and their performance on each subset is also comparable to each other. Therefore, their overall performance is very similar as we observed in Fig. 6.

## 5. RELATED WORK

It is important to point out the differences between our task – query suggestion – and query reformulation. Query reformulation techniques [10, 23] usually substitute some of the query words or phrases, delete them or add new words to the query. Query suggestion, on the other hand, treats queries as a whole and identify suggestions for them from past observed queries. As a results, suggestions are more "natural" than reformulated queries since they are "real" queries. Consequently, suggestions are more suitable for presenting to users as alternatives.

There are quite a few techniques that have been proposed for query suggestion. Beeferman and Berger [2] use agglomerative clustering approach to group similar queries together where the similarity of two queries is defined to be the fraction of common clicked documents they have. Instead of using only click information, Baeza-Yates et al. [1] also takes into the account the content of the page. Each query is represented as term vector aggregated over all clicked documents and these term vectors are clustered using k-mean. Wen et al. [24] use a similar approach but consider query-query content similarity and document-document similarity defined in some taxonomy. Our approach, which is based on a random walk, is different to those because it does not use content similarity or any taxonomy. The approach of Beeferman and Berger is the most similar to ours in that regard, but it requires the computation of pair-wise similarity between any pair of queries, which is less efficient.

Our approach, in fact, is inspired by the one proposed by Mei and Church [16]. They use the notion of hitting time, which is the expected number of steps that a random walk hits a particular query, to rank suggestions. The general idea is suggestions are better if they have shorter expected number of steps of reaching the initial query. While this is a *backward* walk model, we conduct a *forward* walk.

With the emergence of parallel processing frameworks, MapReduce has been used to speed up many tasks [14, 3]. The MapReduce implementation of graph algorithms such as PageRank has been proposed [15]. However, this is not the case for random walk even though it is a very popular algorithm with many applications including ranking documents [4], smoothing click-through data [9] and label propagation [11, 13, 22]. It is worth mentioning that PageRank is a query-independent random walk whereas random walks for query suggestion are query-dependent, which has higher complexity. To the best of our knowledge, an implementation for random walks in the MapReduce framework has not been presented before.

Anchor text has been noted as useful by many researchers [19, 8, 17, 25]. However the most relevant to our study which aims to use anchor text as a substitute for query logs is the work by Dang and Croft [6]. The main difference is that they use anchor text for query reformulation whereas we address the task of query suggestion. In addition, our anchor log is significantly larger than theirs, which was based on the TREC GOV2 collection.

## 6. CONCLUSIONS

In this paper, we construct a simulated query log from anchor text extracted from the ClueWeb-09 collection. We compare this log with a real query log (the MSN log) in terms of the retrieval performance of suggested queries they generate via the random walk procedure. Our main finding is that our anchor log is at least as effective as the MSN log for query suggestion. In fact, the suggestions it provides are significantly better than those obtained with the MSN log on the query set from TREC Web Track 2009. Part of the reason for this may be that the anchor log is larger, and thus has better coverage than the MSN log. This, in fact, further confirms the potential of anchor text which is freely available in vast amount.

In addition, we present an implementation of the popular

random walk procedure in the MapReduce framework. We have also shown that it is not necessary to maintain all new reachable vertices at each step of walk. Doing so not only increases the computational expense, it diffuses the walk away from good suggestions. Instead, we can only keep the top-$K$ new discovered vertices in each step. This substancially cuts down the number of vertices that need to be considered while still performing comparably to the standard procedure. Consequently, each iteration of the walk on a graph with millions of vertices can be done in a matter of minutes.

Currently, there are no known techniques that can consistently provide a single best suggestion for users' queries. Therefore, we evaluated a list of suggestions by judging the best one among them. Interestingly, both the anchor log and the MSN log can provide suggestions that are significantly better than the original query. We see that these high quality suggestions occur as soon as the third position in ranked list of suggestions. We believe that reranking approaches [12] can help push them to the top position.

Even though the *forward* random walk model we implemented in this paper is probably the simplest one, it performs very well in our experiments. Craswell and Szummer [4] achieve better results with *backward* walks than with *forward* walks, suggesting we can benefit more with other variants. We intend to evaluate these models in more detail.

Morever, we observed that the two logs provide different suggestions to many of the queries. In the future, we will also investigate the possibility of combining anchor text with query logs.

## 7. ACKNOWLEDGMENTS

This work was supported in part by the Center for Intelligent Information Retrieval and in part by NSF CLUE IIS-0844226. Any opinions, findings and conclusions or recommendations expressed in this material are the authors' and do not necessarily reflect those of the sponsor.

## 8. REFERENCES

[1] R. Baeza-Yates, C. Hurtado and M. Mendoza. Query recommendation using query logs in search engines. In *The ClustWeb Workshop*, 2004.

[2] D. Beeferman and A. Berger. Agglomerative clustering of a search engine query log. In *Proceedings of KDD*, 2000.

[3] M. Cartright, J. Allan, V. Lavrenko and A. McGregor. Fast Query Expansion Using Approximations of Relevance Models. In *Proceedings of CIKM*, pages 1573-1576, 2010.

[4] Nick Craswell and Martin Szummer. Random walks on the click graph. In *Proceedings of SIGIR*, 2007.

[5] W.B. Croft, D. Metzler, and T. Strohman. *Search Engines: Information Retrieval in Practice*. Addison-Wesley, 2009.

[6] V. Dang and W.B. Croft. Query Reformulation Using Anchor Text. In *Proc. of WSDM*, pages 41-50, 2010.

[7] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proceedings of OSDI*, pages 137-150, 2004

[8] N. Eiron and K.S. McCurley. Analysis of anchor text for web search. In *Proceedings of SIGIR*, pages 459-460, 2003.

[9] J. Gao, W. Yuan, X. Li, K. Deng and J.Y. Nie. Smoothing clickthrough data for web search ranking. In *Proceedings of SIGIR*, 2009.

[10] R. Jones, B. Rey and O. Madani. Generating Query Substitutions. In *Proceedings of WWW*, pages 387-396, 2006.

[11] A. Joshi, R. Kumar, B. Reed and A. Tomkins. Anchor-based Proximity Measures. In *Proceedings of WWW*, 2007.

[12] G. Kumaran and V.R. Carvalho. Reducing long queries using query quality predictors. In *Proceedings of SIGIR*, pages 564-571, 2009.

[13] X. Li, Y. Wang and A. Acero. Learning query intent from regularized click graphs. In *Proceedings of SIGIR*, 2008.

[14] J. Lin. Brute force and indexed approaches to pairwise document similarity comparisons with MapReduce. In *Proceedings of SIGIR*, pages 155-162, 2009.

[15] J. Lin and M. Schatz. Design patterns for efficient graph algorithms in MapReduce. In *Proceedings of MLG*, pages 78-85, 2010.

[16] Q. Mei, D. Zhou and K. Church. Query suggestion using hitting time. In *Proceedings of CIKM*, 2008.

[17] D. Metzler, J. Novak, H. Cui, and S. Reddy. Building enriched document representations using aggregated anchor text. In *Proceedings of SIGIR*, pages 219-226, 2009.

[18] *Proceedings of the 2009 workshop on Web Search Click Data*, Barcelona, Spain. ACM New York, NY, USA, 2009.

[19] R. Nallapati, W.B. Croft and J. Allan. Relevant query feedback in statistical language modeling. In *Proceedings of CIKM*, pages 560-563, 2003.

[20] F. Peng, N. Ahmed, X. Li, and Y. Lu. Context sensitive stemming for web search. In *Proceedings of SIGIR*, pages 639-646, 2007.

[21] T. Strohman. Efficient Processing of Complex Features for Information Retrieval. PhD thesis, University of Massachusetts Amherst, University of Massachusetts Amherst, December 2007.

[22] M. Szummer and N. Craswell. Behavioral classification on the click graph. In *Proceedings of WWW*, 2008.

[23] X. Wang and C. Zhai. Mining term association patterns from search logs for effective query reformulation. In *Proceedings of CIKM*, pages 479-488, 2008.

[24] J. Wen, J.Y. Nie and H.J. Zhang. Clustering user queries of a search engine. In *Proceedings of WWW*, 2001.

[25] X. Yi and J. Allan. A Content based Approach for Discovering Missing Anchor Text for Web Search. In *Proceedings of SIGIR*, pages 427-434, 2010.