Transfer Understanding from Head Queries to Tail Queries

Yangqiu Song
UIUC
yqsong@illinois.edu

Haixun Wang
Google Research
haixun@google.com

Weizhu Chen
Microsoft
wzchen@microsoft.com

Shusen Wang
Zhejiang University
wss@zju.edu.cn

ABSTRACT

One of the biggest challenges of commercial search engines is how to handle tail queries, or queries that occur very infrequently. Frequent queries, also known as head queries, are easier to handle largely because their intents are evidenced by abundant click-through data (query logs). Tail queries have little historical data to rely on, which makes them difficult to be learned by ranking algorithms. In this paper, we leverage knowledge from two resources to fill the gap. The first is a general knowledgebase containing different granularities of concepts automatically harnessed from the Web. The second is the click-through data for head queries. From the click-through data, we obtain an understanding of queries that trigger clicks. Then, we show that by extracting single or multi-word expressions from both head and tail queries and mapping them to a common concept space defined by the knowledgebase, we are able to transfer the click information of the head queries to the tail queries. To validate our approach, we conduct large scale experiments on two real data sets. One is a mixture of head and tail queries, and the other contains pure tail queries. We show that our approach effectively improves tail query search relevance.

Categories and Subject Descriptors
I.2.0 [**Artificial Intelligence**]: General; H.2.8 [**Database Management**]: Database applications–*Data mining*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval.

General Terms
Algorithms; Experimentation

Keywords
Knowledgebase; Short Text Conceptualization; Query and URL Understanding; Search Relevance.

1. INTRODUCTION

Web search queries follow a heavy tailed Zipf distribution [31]. We classify queries into head queries and tail queries [8]. Head queries, also known as frequent queries, are associated with rich historical click information. This enables search engines to utilize statistical models to predict the URL's click-through rate as search relevance [19, 28]. Tail queries, also known as rare queries, do not have much user click information. Unfortunately, tail queries constitute the majority of unique queries submitted by users in their day-to-day use of search engines. It is very valuable but a big challenge for search engines to improve search relevance for tail queries.

CIKM'14, November 3–7, 2014, Shanghai, China.

http://dx.doi.org/10.1145/2661829.2662078 .

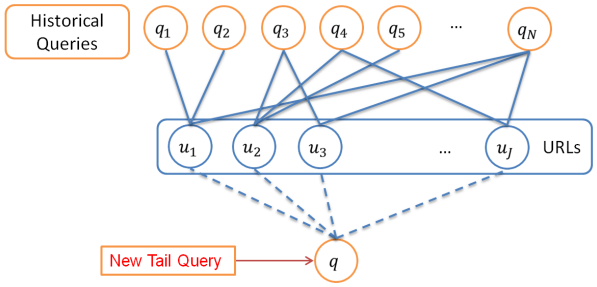


Figure 1: An example of tail query search solution. q is a new coming tail query that has not seen before. $q_1, \cdots, q_N$ are the historical queries triggered the clicks of URLs $u_1, \ldots, u_J$.

For example, we can evaluate query and document similarity in the TFIDF (term frequency, inverse document frequency) vector space as a relevance measure. This leads to a list of candidate URLs. As shown in Fig. 1, for a new tail query q, we obtain candidate URLs $u_1, \ldots, u_J$ by query document similarity. However, many of them may have low relevance even though they have high similarity, because simple content similarities using TFIDF may be different from the users' intents as clicks can reveal. For example, when a user searches a very rare car insurance, e.g., "geely king kong car insurance," he really means the insurance of the particular car rather than the car itself or even the character King Kong who damaged cars in a movie. Thus, the challenge is *how to better estimate the intents of tail queries?*

Several approaches have tried to use search logs to improve search relevance for tail queries [12, 13, 16, 29, 34]. For example, we can use the set of queries that historically triggered clicks of a URL as the description of the URL, since if a query triggers the click of a URL, we regard that the URL satisfies the intent of the user. An example of this approach is shown in Fig. 1. Given the goal is to rank URLs $u_1, \cdots, u_J$ by their relevance to q, the relevance can be converted to a similarity measurement between q and $q_1, \cdots, q_N$ in this model. Since the link between q_i and u_j can be weighted by the number of clicks of u_j triggered by q_i, the user intent of tail query q can be estimated by the weighted average of the historical queries' intents.

The above approach of converting the relevance problem into measuring click-weighted query similarities could be useful in prac-

tice. Nonetheless, there is still some drawback, since a query is just a piece of short text, and there is not sufficient contextual information to compare the semantics of two short texts. For example, "seattle hotel jobs" and "seattle hotel" are two queries, and they have relatively smaller edit distance than the distance between queries "mars planet" and "venus." However, the latter two queries are more semantically similar and the former two represent different meanings. Therefore, the string similarity or bag-of-words based similarity for queries are not enough to evaluate the semantic similarity. Several approaches tried to use latent topic models [13] or deep learning model [17] to extract the latent or hierarchical semantics of queries. In practice, they are slow to train and test. Thus, they are only applied to query-title similarities but not historical queries [17, 13]. However, we will show that, considering both the semantics about historical queries and the user clicks can be much more useful.

In this paper, instead of evaluating query similarity base on their surface forms or latent topics, we use explicit semantic representation of queries. We then evaluate the query similarity in the semantic space. For example, we know "seattle hotel jobs" is different from "seattle hotel," since "jobs" is mapped to concepts different from the concepts "seattle" and "hotel" are mapped to. Moreover, both "mars planet" and "venus" can be mapped to the concept *planet* or *planet from the Sun*. In this case, the similarity between these two queries is much larger than the similarity evaluated based on the strings. Thus, if we can correctly parse queries into sub-phrases and identify their concepts, we can relate queries which are different on their surfaces. Then the challenges are how to parse queries into single or multi-word expressions automatically, how to create the concept space, and how to map queries to the concept space. The concept space should be large enough to have broad coverage of queries and should be with different granularities of semantics to describe all kinds of queries.

To obtain the concepts, we use a probabilistic knowledgebase, Probase [36]. Probase contains several millions of concepts and related instances and attributes names, which are extracted from a text corpus of 1.68 billion Web pages [36]. It covers more than 80% of Web queries evaluated on 50-million randomly selected queries from a real-world search engine [36]. Since the concepts, instances and attributes are single or multi-word expressions, they enable us to detect terms in queries which are more semantically meaningful. Then we extend the technique called "short text conceptualization" [32] to capture multiple topics inside a query using simple but effective clustering, and consequently provide a general way to map both queries and URLs to the concept space.

Our contributions are the following two key components that together enable us to better characterize tail queries with semantics for search relevance.

- First, we employ large scale, probabilistic knowledge about concepts with different granularities in query understanding. Using the knowledge, we extract single or multi-word expressions in queries, map queries (both head and tail queries, with single or multiple semantic topics) to the concept space, and compare query similarity in the concept space.

- Second, given the representation of URL as historically clicked queries, we measure the semantic similarity between a tail query and a URL based on both the concept similarity and the popularity of the queries that triggered the clicks of the URL. Then the similarity depends on the number of clicks, and thus emphasizes on head queries since they are more popular. In this case, we transfer the understanding of users' intent knowledge from head queries to tail queries.

2. RELATED WORKS

In this section, we briefly review the methods of query expansion and URL representation for improving search relevance.

2.1 Query Expansion

Query expansion approaches mainly fall into two categories: corpus-based and general-purpose knowledge-based. The corpus-based approaches use snippets [37], anchor texts [21] or search log session data [30] to cluster historical queries. The general-purpose knowledge-based approaches involve different knoweldgebases, including WordNet [25], Wikipedia [24], ConceptNet [14, 20] or the combination of multiple knowledgebases [6, 15, 26].

However, corpus-based and general-purpose knowledge-based methods are insufficient for the tail query problem. Given a tail query, even if we can expand it correctly, the expanded queries might still be tail queries. In addition, one should be very careful using the expanded queries, because search results are sensitive to terms. Therefore, in practice, query expansion is often used as a preprocessing technique to find synonyms, morphological forms of words or to perform spell checking. Moreover, query expansion can be used as a tool to do query suggestion after the user submits a query.

2.2 URL Representation

Since URLs cannot be directly compared with queries, they are usually represented by related texts. Besides the document content of a URL, real-world search engines also seek other stronger signals to help improve the similarity measure between queries and URLs to improve relevance. There are two main types of fields used to construct the representation of URLs. The source generated by the author of a Web page is called the content field, while the source generated by the other authors is called the popularity field [29, 34]. Content fields include page body, title, and the URL text. Unlike content fields, popularity fields can be the anchor texts, and the queries that trigger the clicks, which can better capture user's intent information [29, 34].

Inspired by the initial work proposed in [29, 34] to use popularity fields to help search, there have been several advanced statistical methods developed [12, 13, 16]. Huang et al. [16] demonstrated that phrases (or multi-word expressions) can be more useful than words for modeling the similarity between queries and URL representations, using an n-gram language model. Gao et al. also extended the word-level translation model [2] to the phrase-level translation [12], and showed that phrases with unrestricted lengths can help improve search relevance. Language models and translation models are very powerful tools, however, they "do not map different terms into the semantic clusters but only learn the mapping relationships directly between a term in a document and a term in a query" [13]. Therefore, sometimes the semantically related queries are not identified using such models. Thus, Gao et al. [13] introduced a latent topic model to improve the bag-words-model. They claims that latent topic models [3] can effectively capture the semantic meaning of queries and documents. More recently, Huang et al. improved the latent model by learning a deep structure to incorporate hierarchical semantic information [17]. However, they are still word based instead of phrase based. Moreover, training such latent models is time consuming, and thus these latent models are only applied to the title field. In addition, latent models are not explicitly understandable for human.

Our work also addresses the problem with single or multi-word expressions, but we instead use a Web-scale knowledgebase. Our knowledgebase contains concepts, attributes, instances, and their relationships with good precision and coverage of Web queries [36]. Thus, the multi-word expressions extracted using the knowledge-

base are more meaningful. Moreover, the knowledgebase contains a set of taxonomic concepts with different granularities, which reflect different levels of semantics. Thus, mapping the terms (extracted single or multi-word expressions) to the explicit concept space can naturally incorporate the hierarchical semantic structure as deep learning model [17]. Egozi et al. [9] have demonstrated that mapping queries to the explicit concept space can significantly improve the search relevance results, however, they haven't incorporated the historical click information or other popularity fields. In this work, we combine the benefits of explicit semantic analysis which can be understood by human for debugging, and the popularity fields that reflect users' intents, and develop a new model. Our model can transfer the understanding of queries from head to tail by mapping queries to the same concept space, and leveraging the click information.

3. EXTERNAL KNOWLEDGE

In this section, we briefly introduce how we acquire relevant knowledge, and the basic idea of conceptualization. There are many knowledgebases including manually built ones such as WordNet [11], Open Directory Project (ODP)[1], Wikipedia[2], Cyc [22], and Freebase [4], and automatically constructed ones such as KnowItAll [10], TextRunner [1], WikiTaxonomy [27], YAGO [33], NELL [5], and Probase [36][3].

In this paper, we use Probase as the knowledge source that fills the gap between sparse input and understanding. Probase contains more than two million concepts learned iteratively from 1.68 billion web pages. The taxonomy is probabilistic, which means every claim in Probase is associated with some probabilities that model typicality, correctness, ambiguity, and other characteristics. The probabilities are derived from evidences found in Web data, search log data, and other available data. The knowledge we acquire centers around two relationships that are associated with concepts, namely, the *isA* relationship, e.g., *China isA emerging market*, and the *isAttributeOf* relationship, e.g., *population isAttributeOf country*. We acquire such knowledge through iterative information extraction from web data. Specifically, to obtain *isA* relationships, we use Hearst patterns and other syntactic patterns. For instance, from a sentence "... European artists *such as* Pablo Picasso ...," we obtain *Pablo Picasso isA European artist*. To obtain *isAttributeOf* relationships, we consider sentences such as "*What is* the population *of* China?", from which we obtain *population isAttributeOf China*. We perform sophisticated inference to clean and enrich the knowledge acquired from the syntactic patterns [36].

In order for the knowledge to be useful for conceptualization or understanding, it needs to satisfy two requirements. First, the knowledge should have a broad coverage. People use millions of concepts (e.g., *emerging market*, *European artist*, etc.) in their communication and a query can contain any of those concepts. Thus, we need a general knowledgebase that can cover as many concepts as possible. Existing knowledgebases [1, 4, 10, 11, 22, 27, 33] contain a limited number of concepts, which limits their power in interpreting the natural language. Probase covers more than 80% of Web queries evaluated on 50 millions randomly selected queries from a real-world search engine [36], which means it can cover almost all the head queries and most of the tail queries. This lays the foundation for understanding the human generated queries which express subtle meanings and intents.

Second, in order to support inference, the acquired knowledge in Probase provides probabilistic information. For example, the probability $P(instance|concept)$ tells us how typical the *instance*[4] is in the given *concept*. Intuitively, knowing that both "robin" and "penguin" are *birds* is not enough. We need to know robin is a much more typical bird than penguin, that is, when people talk about birds, it is more likely that they are thinking about robins than penguins. Such information is essential for understanding the intent behind a short text. Besides $P(instance|concept)$, we also obtain $P(concept|instance)$, $P(concept|attribute)$, and $P(attribute|concept)$. These values are obtained during the information extraction process, for example:

$$P(instance|concept) = \frac{n(instance, concept)}{n(concept)} \quad (1)$$

where $n(instance, concept)$ denotes the number of times *instance* and *concept* appear in the same sentence in a corpus, and $n(concept)$ is the frequency of the *concept*.

4. QUERY AND URL CONCEPTUALIZATION

We use external knowledgebases for short text understanding. We first map queries (head queries and tail queries) to the concept space (Section 4.1) and then we expand URLs by adding textual context so that we can map URLs to the concept space as well (Section 4.2).

4.1 Query Conceptualization

Given a query, which consists of a set of terms (single or multi-word expressions), we want to infer the concepts from the query. We call the process conceptualization. Fig. 2 shows two examples of conceptualization. We can see that from terms "microsoft" and "apple," we get *company* related concepts, while for terms "pear" and "apple," we get *fruit* related concepts. However, queries are complicated. For example, they often contain multiple topics (e.g., a query such as "Obama's real-estate policy"). In this section, we describe the techniques of conceptualizing any short text.

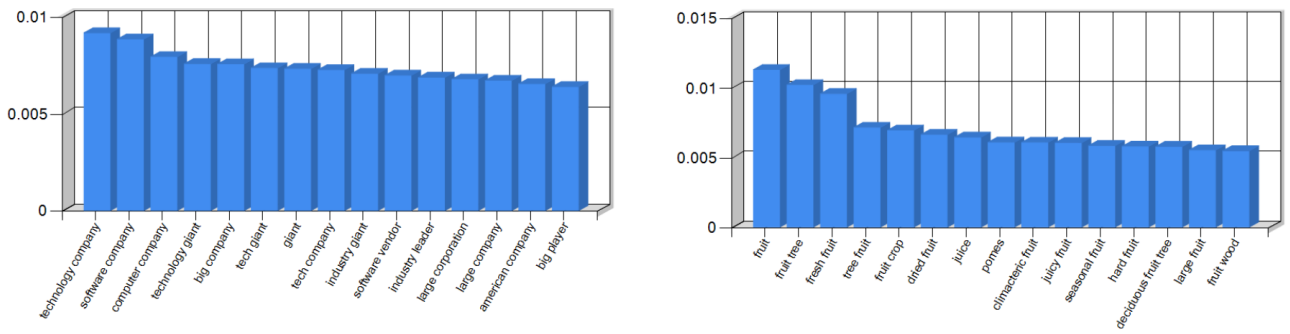
Parsing

Before mapping a query to concepts we want to parse it meaningfully into a set of terms that exist in our knowledgebase. For example, for query "disposable chopstick manufacturer," we obtain {"disposable chopstick," "manufactures"}, where both are the longest terms in the knowledgebase that appear in the query (the knowledgebase also contains "chopstick" as a term, but we assume it does not contain "disposable chopstick manufacturer" as a single term). Similarly, there are many cases where we should make decisions to find the most meaningful terms in the queries.

In general, suppose our knowledgebase contains a set of terms $\{t_1, \cdots, t_V\}$, where term t can be an instance e, an attribute a or a concept c. A piece of short text consists of a sequence of words. A term occurs as a contiguous sequence of words in a short text. We use $l(t)$ to denote t's length. For example, l("disposable chopstick") = 2. We use $n(t)$ to denote the number of concepts t corresponds with. For example, if t = "apple" corresponds with the concepts of *fruit* and *company* only, then $n(t) = 2$. A term may correspond to a large number of concepts. As an example, we have n("driving school") = 18 and n("San Diego") = 327.

Our parsing method tries to find the *Longest Covering Term (LCT)* of short text (see Algorithm 1). We say a term t covers a word w if both t and w appear in the short text and t contains w. For example, "disposable chopstick" covers "chopstick." The algorithm ensures that every word in the short text is covered by a term (if possible). We denote $t_i \geq t_k$ if $l(t_i) > l(t_k)$, or if $n(t_i) \geq n(t_k)$ and $l(t_i) = l(t_k)$.

[1] http://www.dmoz.org/
[2] http://www.wikipedia.org/
[3] http://research.microsoft.com/probase/
[4] Here, we use *instance* to denote a sub-concept or an entity.



(a) Conceptualization of "microsoft" and "apple."

(b) Conceptualization of "pear" and "apple."

Figure 2: Conceptualization of terms using [32]. Y-axis represents the significance weight for each concept. Larger weights means more important the concept is to represent the text.

Algorithm 1 The Longest Covering Term Algorithm

Input: a short text;
Best covering term set $\mathcal{T} \leftarrow \emptyset$;
Break the text into a sequence of words $(w_1, \cdots, w_N)$;
Term set $\mathcal{W} \leftarrow$ detect all terms from $(w_1, \cdots, w_N)$;
for $i = 1$ to N **do**
    Term set $\mathcal{W}' \leftarrow$ all terms in $\mathcal{W}$ that covers w_i;
    $t \leftarrow$ best term in $\mathcal{W}'$ w.r.t term comparison $\succeq$:
        $t_i \succeq t_k$ if $l(t_i) > l(t_k)$, or if $n(t_i) \geq n(t_k)$ and $n(t_i) = n(t_k)$
    $\mathcal{T} \leftarrow \mathcal{T} \bigcup \{t\}$;
end for
Output: best covering terms $\mathcal{T}$.

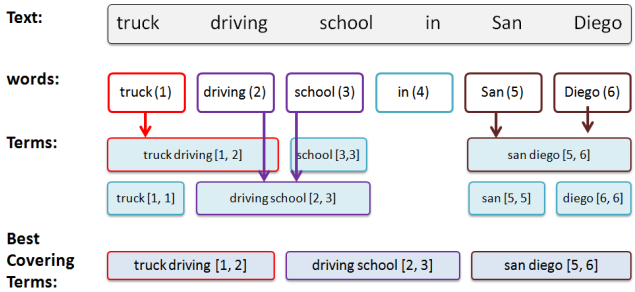


Figure 3: The first line "truck driving school in San Diego" is a given short text, which contains 6 words; these 6 words make up 7 terms. Each word is labeled with its position in the text, and each term is labeled with its range. Each word chooses the longest covering term, as denoted by an arrow in the figure. Note that the word "driving" has two covering terms of the same length, "driving school" is chosen since it corresponds to more concepts than "truck driving."

When multiple terms are detected, we choose the best term (with respect to "$\succeq$") to cover each word. An example is shown in Fig. 3.

Single-topic Conceptualization

For now, let us assume all terms in a query reflect the same topic (e.g., "microsoft" and "apple" are *companies*, while "microsoft windows," and "windows 8' refer to *operating systems*), and that our goal is to derive that topic. To do this, we identify the top K candidate concepts $C = \{c_k\}$ ranked by their likelihood from a set of terms of unknown types $\mathcal{T} = \{t_1, \ldots, t_L\}$. Here, the type of a term can be either an instance or an attribute. Note that the same term can serve as an instance as well as an attribute. For example, "population" can be an attribute of *country*, but it can also be an instance of *geographical data*. However, it is rare that a term is both an instance and an attribute of the same concept.

We introduce an auxiliary variable z_l to indicate the type of term t_l. Specifically, $z_l = 1$ if t_l is an instance, and $z_l = 0$ if t_l is an attribute. Given that the knowledgebase contains noise, we handle the logic in a discriminative manner. We introduce a noisy-or model to first infer the probability $P(c_k|t_l)$:

$$P(c_k|t_l) = 1 - (1 - P(c_k|t_l, z_l = 1))(1 - P(c_k|t_l, z_l = 0)), \quad (2)$$

where term t_l invokes concept c_k if it is an instance of c_k, or an attribute of c_k. Here, we have

$$P(c_k|t_l, z_l = 1) = P(c_k|e_l) = P(c_k, e_l)/P(e_l), \quad (3)$$

where term t_l is regarded as an instance e_l, and

$$P(c_k|t_l, z_l = 0) = P(c_k|a_l) = \sum_{i:e_i \in E} P(c_k|e_i)P(e_i|a_l), \quad (4)$$

where term t_l is regarded as an attribute a_l, and E is the set of instances that are related to attribute a_l and concept c_k. Then, using the naive Bayes rule, we derive the concept posterior given a set of terms by:

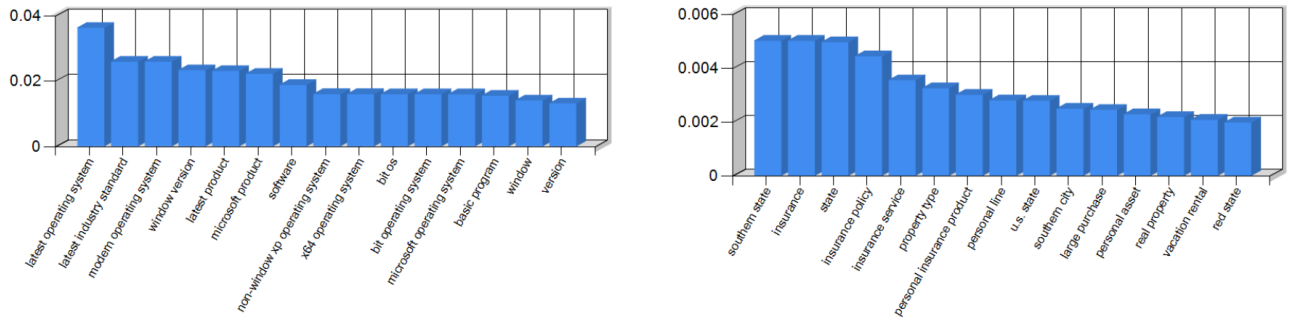
$$P(c_k|\mathcal{T}) \propto P(c_k) \prod_l^L P(t_l|c_k) \propto \frac{\prod_l P(c_k|t_l)}{P(c_k)^{L-1}}, \quad (5)$$

where $P(c_k|t_l)$ is given by Eq. (2) [32].

Multi-topic Conceptualization

The parsing results of many queries consist of terms that reflect more than one topic. For example, for query "disposable chopstick manufacturer," we get {"disposable chopstick," "manufacturer"}, and for query "alabama home insurance," we get {"alabama," "home insurance"}. The previous conceptualization method performs a joint inference on the terms. The Bayesian rule in Eq. (5) tends to emphasize the common concepts belonging to the terms. Even after using smoothing techniques in naive Bayes [32], it may still lead to general, vague, and even meaningless concepts such as *object*, *item*, *thing*, etc.

We capture multiple topics in a query through term clustering. We first represent each term t_i as a concept vector $\mathbf{c} = (c_1, \ldots, c_M) \in \mathcal{R}^M$ where M is the total number of concepts in the knowledgebase. In practice, we only retrieve the top K concepts for each term and thus $\mathbf{c}_i$ is a sparse vector. Then we formulate the term relationship as a connected weighted graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ is the vertex set and $\mathcal{E}$ is the edge set. We define the terms detected in a query as vertices in the graph, and define weights on the edges between them



(a) Conceptualization of "microsoft windows 8."

(b) Conceptualization of "alabama home insurance."

Figure 4: Conceptualization with clustering. Y-axis represents the significance weight for each concept. Larger weights means more important the concept is to represent the text.

using cosine similarities between concept vectors. More specifically, we define the weight s_{ij} between t_i and t_j as:

$$s_{ij} = \text{cosine}(\mathbf{c}_i, \mathbf{c}_j) = \frac{\mathbf{c}_i^T \mathbf{c}_i}{\|\mathbf{c}_i\| \cdot \|\mathbf{c}_j\|}. \quad (6)$$

Then, we devise a clustering algorithm to find connected sub-graphs after filtering out edges whose weights are below a threshold. We use a pre-defined threshold to filter out non-significant edges. Thus, terms that are not semantically related will not be clustered together. Then we discover connected sub-graphs using a simple graph traverse algorithm, and all the terms that are connected to each other will be grouped together. The time complexity is in quadratic order of the number of terms detected in a query $O(L^2)$, because we compute pairwise similarity in the beginning of the algorithm. In general, a query has very small number of terms. Therefore, the computational cost for clustering is not significant.

Given the term clusters, we then use Eq. (5) to conceptualize each cluster of terms. In this way, the common concepts of a set of related terms will be ranked higher than the individual ones. For example, as [32] pointed, given "China" and "Japan," the top concepts will contain *Asian country* and *eastern country*. However, given "China" and "Russia," the top concepts will contain *emerging market* and *developing country*. Then for each cluster r discovered by simple graph cut, we represent it as a concept vector $\mathbf{c}_q^r$, which is the concept vector of rth cluster of query q, using Eq. (5). Then the conceptualization of a query is given by:

$$\mathbf{c}_q = \frac{1}{L} \sum l_r \cdot \mathbf{c}_q^r, \quad (7)$$

where L is the number of terms detected in a query, and l_r is the number of terms grouped into cluster r. Now the conceptualization result of a query is a weighted mixture of multiple topics, where each topic is described by a set of related concepts. When we compare the semantic relationship between this query and the other queries, different topics will contribute differently.

Fig. 4 shows two examples of conceptualization results. For the query "microsoft window 8," our algorithm automatically determines that there is only one topic about operating systems in that query. For the query "alabama home insurance," our algorithm identifies that there are two topics, and balance these two topics to get better concept distribution.

4.2 URL Conceptualization

In web search, given a query q, we want to evaluate its relevance to a set of URLs $u_1, u_2, \cdots$. Figure 5 illustrates the setting of the problem. Using the techniques described in Section 4.1, we map q to a set of concepts $c_1, c_2, \cdots$. The same query q is also mapped to a set of URLs $u_1, u_2, \cdots$. Our goal is thus to establish the relationship between the concepts and the URLs, so that knowledge from the URLs can be channeled by the concepts to the tail query. In this section, we show how we can introduce another layer in the figure (shown as the top layer in the figure), through which the knowledge in the concepts can be used to filter and rerank the URLs.

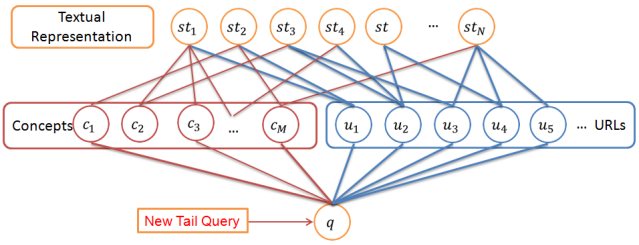


Figure 5: Knowledge from concepts goes through an additional layer to act upon a set of URLs and evaluate their relevance. st_i **here means short text, which can be clicked query, anchor text, or title of URL.**

Textual Representations of a URL

URLs by themselves do not contain much information. They are not natural language texts and the conceptualization mechanisms we developed cannot be applied to URLs directly. In our approach, we introduce an additional layer as shown in Fig. 5: the textual representation of the URLs. Our goal is to convert URLs to a textual representation, and then map the textual representation to the concept space. The question is then, what are the appropriate textual representations of URLs. There are a few options. In this paper, we discuss three of them: clicked queries, anchor texts, and URL titles.

- **Clicked queries:** For a query q, the search engine returns a set of URLs. Assume u is a URL clicked by the user. We call q a clicked query of u. We use the set of clicked queries as a textual representation of u. The motivation is that if the URL is clicked by a query, we have reasons to believe that the URL satisfies the intent of the user's query. For example, for URL *http://www.monster.com*, representative queries that trigger clicks on the URL include "monster.com," "hotjobs.com," "jobs," "career monster," etc. These queries form a good description of the website the URL points to. More examples of clicked queries are listed in Table 1, where each query is weighted by the number of clicks.

Table 1: Examples of URLs and the clicked queries.

URL	Representative queries (click count)
http://www.monster.com	monster.com (67507); monster jobs (8155); job search (2654); jobs (2332); hotjobs (1271); ...
http://www.petsuppliesplus.com	pet supplies plus (19102); pet supplies (495); Pet Supplies plus Stores (46); ...
http://www.dmv.ca.gov/pubs/interactive/tdrive/exam.htm	Driver License Test (530); dmv california practice test (388); dmv written test questions and answers (45); ...
http://abcnews.go.com/US/Story?id=2721089&page=1	520 bridge motorcycle tolls (1); what the families of jumpers say about The Bridge (1)
http://chemistry.about.com/cs/chemistry101/a/aa071503a.htm	reactions in aqueous solutions (10); Ions in an Aqueous Solution (1); what type of reaction is Zn+HCl (1); ...
http://www.northerntrust.com/pws/jsp/display2.jsp?XML=pages/nt/0601/1138283678319_6.xml&TYPE=interior&dc=30	northern trust economic research (32); northern trust economics (2); northern trust paul kasriel (1)
http://baucus.senate.gov	senator of montana (1)
http://download.oracle.com/docs/cd/B28359_01/server.111/b28310/schema002.htm	enable resumable timeout sql (1)

Table 2: Examples of URLs and the anchor texts.

URL	Representative anchor text (count)
http://www.the-master-cleanse.net	www the master cleanse net (222); master cleanse detox (162); the grasp cleanse net (145); ...
http://www.usairways.com	us airways mainline (258); us airways home page book travel airlines official site (114); us airways inc us (179); ...
http://jetblue.com	jet blue airways home page (209); jetblue airline tickets flights and (207); jet blue site (138); online offering (93); ...
http://kidscluster.com	"kids cluster (1115); newsletter for upcoming buying group of 7000 retailers (243); visit (145)
http://www.djbooth.net	new hip hop songs music charts interviews (99); music downloads hip hop .net (74); new hip hop mixtapes albums (90); ...
http://www.workhappynow.com/2009/07/why-dont-people-laugh-work/	fear of laughing in the office (19); why people don t laugh at work (24); why don t people laugh at work (11); work happy now (9);
http://allrecipes.com/Recipe/Fresh-Fruit-Flan/Detail.aspx	fresh fruit flan allrecipes (93); http allrecipes com recipe fresh fruit flan detail aspx src rss (93); fresh fruit flan (132); ...

- **Anchor texts:** The anchor text is the visible, clickable text in a hyperlink. For each URL u, we collect the anchor texts in the hyperlinks where u is referenced. We then use the set of unique anchor texts as the textual representation of u. The motivation is that an anchor text gives relevant descriptive or contextual information about the destination of the hyperlink [34]. Furthermore, the more popular an anchor text is, the more likely it truly represents URL u. More examples are shown in Table 2, where each anchor text is associated with the number of times it is used as its weight.

- **URL titles:** For each URL, we also obtain the title of its destination page [34]. Unlike clicked queries and anchor texts, each URL corresponds to only one title. This also means the link between the title and the URL has a unit weight. Another difference is that the title is the description given by the author of the web page, while clicked queries and anchor texts represent what other people think about the page. Because of the differences, conceptualization based on titles does not bring as significant improvement as anchor texts and click queries, as shown by our experimental results.

Conceptualization

We add textual representation for URLs as an additional layer in Fig. 5. Whether the layer consists of a set of clicked queries, or a set of anchor texts, or a title, we can conceptualize them as short texts. This enables us to map the URLs to concepts indirectly through the added layer.

Let u denote a URL, and let $\{st_1, \ldots, st_N\}$ denote its textual representation, where st_i is short text (a clicked query, an anchor text, or a title). We map u to a concept vector $\mathbf{c}_u$ by

$$\mathbf{c}_u = \frac{1}{\sum w_{st_i}} \sum_{i=1}^{N} w_{st_i} \mathbf{c}_{st_i}. \quad (8)$$

Here, w_{st_i} is the score of short text st_i, and it is defined as $w_{st_i} = \log[\text{count}(st_i) + 1]$ where $\text{count}(st_i)$ is the frequency of short text st_i co-occurring with URL u (that is, how many times a unique anchor text is used when referencing u in a hypertext, or how many times u is clicked in response to a query). The intuition is to weight the concepts with the count. For example, if a query triggered more clicks of that URL, the concepts describing that query will be weighted more importantly when computing the concepts of the URL.

Now, we have represented each URL as a vector of concepts. Then the queries and the URLs are comparable in the same space. For historical queries, we compute the query-URL pairwise similarities and input them to a learning to rank algorithm associated with the other features extracted from the pairs to learn the parameters. For the online search problem, we should compute the conceptualization of URLs offline. This can be handled using an enterprise level MapReduce-type system in days. Given a new query, which may be a tail query we have never seen before, we compute the concepts of that query and compute the similarities between the query and the candidate URLs. Then the similarities are input as features for the learning to rank system for further judgement of relevance.

5. EXPERIMENTS ON REAL DATA

In this section, we present the experimental results conducted on two real data sets, and show the performance of how conceptualization can improve search relevance. The results for relevance are evaluated using the NDCG (normalized discounted cumulative

Table 4: Statistics of anchor text data on two benchmark data sets.

Data set	Mixture	Tail
# Valid URLs	6,913,177	266,121
Anchor char length	20.93±15.98	19.84±15.75
# Anchor words	3.36±2.61	3.23±2.59
# Unique anchor texts for URL	14.47±309.37	68.00±1,460.93

Table 5: Statistics of query clicks over Click 5M data on the two benchmark data sets.

Data set	Mixture	Tail
# Valid URLs	7,230,841	296,400
Query char length	13.01±9.44	10.39±6.27
# Query words	2.07±1.68	1.48±1.01
# Query clicks for each URL	581.71±182,084.84	8,777.85±950,575.36
# Unique query clicks for each URL	62.88±380.51	229.24±1,506.12

Table 3: Statistics of two benchmark data sets.

Data set	Mixture	Tail
# Queries	303,259	12,205
# Total pairs	19,397,159	369,291
# Avg. URLs	63.96	37.24
Query char length	24.51±17.03	30.57±15.45
# Query word count	4.08±3.00	5.18±2.89

gain) [18] score: NDCG@$K = \frac{100}{Z}\sum_{r=1}^{K}\frac{2^{k(r)}-1}{\log(1+r)}$, where $k(r)$ is the relevance label for the document ranked at position r; K is the level that NDCG is computed; and Z is a normalization constant which makes NDCG@K = 100 for the best ranking list. Generally, we compute the average NDCG score over all queries. Obviously, NDCG can be used to evaluate multilevel ranking values in terms of positions for search relevance.

To train a competitive ranking function, we use a gradient boosting tree algorithm [35] which is accepted as a state-of-the-art "learning to rank" algorithm. Additionally, our gradient boosting tree can directly optimize NDCG.

To derive a consistent and trustful experimental report, we conduct our experiment with the two-fold cross validation approach. That means the data are split into training and test sets. Training data are used for learning the ranking function, and test data are used to calculate NDCG results. All the experiments are based on six random splitting trials to compute the average NDCG scores.

5.1 Benchmark Data Sets

In this section, we briefly introduce the two data sets we used. The first data is a mixture of head and tail queries uniformly sampled from search logs. The second data set is elaborately selected which contains only tail queries that appeared only once in one day. A comparison of statistics of two data sets is shown in Table 3. We see that, the average query length of the mixture data is less than the one of tail data. Averagely speaking, tail queries tend to be individual and longer than head queries. The data sets are quite large, because we need to compute all related historical queries and anchor texts associated to the URLs. In practice, we compute the scores using a MapReduce-type mechanism under a distributed computing environment. Our approach can be easily scaled up because we deal with each short text individually using the same knowledgebase. For each pair of query and URL, on average it needs tens of milliseconds on a 2.8GHz PC machine.

5.2 Historical Data Collection

We collect the anchor text data from one day's snapshot of Web pages from a commercial search engine. The data contains billions of URLs and anchor texts with counts, and it takes 286G on disk to store the raw data. We filter out the anchor text data with the URLs contained in the two benchmark data sets. The statistics of the data is described in Table 4.

We see the average word count in anchor text is 3.36 and 3.23 for mixture and tail data sets, namely there are on average about three words for each anchor text. Moreover, the average number of unique anchor text for each URL is 14.47 and 68.00 for mixture and tail data set. It seems that the mixture data set contains both head and tail URLs, however, in the tail data set, there are more head URLs. The variances of numbers of unique anchor text for both data sets are large. This means the distribution of the number has very long tail.

We collect the clicked query data with two data sets to demonstrate the use of historical click information. The first click data contains five months search log, and the second click data contains ten months search log. We name these two sets *Click 5M* and *Click 10M*. They cost 178G and 323G on disk respectively. We also join these two data sets with the two benchmark data sets. The statistical numbers are shown in Tables 5 and 6.

We can see that there are more valid URLs in the 10 months' data than in the 5 months' data. The average query word number is about 2 for mixture benchmark and 1.5 for tail benchmark. This shows that, there are more head queries in the historically clicked query data sets, and the clicked queries joined with the tail benchmark data set tend to be head queries. This is consistent with the conclusion that in the tail data set, there are more head URLs. The average number of query clicks for each URL in tail data is also larger than the one in the mixture data. Moreover, the number of query clicks of each URL in 10 months' data is about two times the number of the 5 months' data. However, the unique number of queries for each URL in the 10 months' data is only one and a half times the number of 5 months' data. This means when we collect the data over a longer time period, the coverage does not grow as fast as the number of clicks grows. This means, head queries and URLs receive more clicks.

5.3 Evaluation and Discussion

We compare our methods with different sources with two ranking baselines. We have four data sources of both mixture and tail queries, i.e., title, anchor, click 5M and click 10M. The title source contains the landing page titles of URLs. The anchor and click data are extracted as the above introduced. The four data sources will have different ranking NDCG results because they do not have the same URL set. From Tables 4, 5 and 6 we see that there are different valid URL numbers for different source and data. Therefore, in the following experiments, we only show how significant con-

Table 6: Statistics of query clicks over Click 10M data on the two benchmark data sets.

Data set	Mixture	Tail
# Valid URLs	7,970,177	325,716
Query char length	13.35±9.37	10.41±6.28
# Query words	2.05±1.66	1.49±1.10
# Query clicks for each URL	1,021.50±351,854.21	15,898.87±1,839,522.70
# Unique query clicks for each URL	97.50±610.28	362.84±2,448.04

Table 7: Comparison of NDCG scores with different single similarity measure for mixture data.

Methods	Title	Anchor	Click 5M	Click 10M
Edit	39.66	41.39	40.59	40.35
Cosine	39.16	41.49	41.39	40.88
Cosine+Edit	**40.38**	**42.32**	**42.26**	41.86
JSScore	38.65	40.63	40.25	39.9
JSScore + Edit	40.29	42.21	42.25	**41.98**
Jaccard	38.59	40.20	39.30	38.93
Jaccard + Edit	39.79	41.47	41.04	40.54

Table 8: Comparison of NDCG scores with different single similarity measure for tail data.

Methods	Title	Anchor	Click 5M	Click 10M
Edit	28.6	27.76	29.9	30.17
Cosine	28.27	27.37	28.89	29.14
Cosine+Edit	**30.03**	29.32	**30.78**	**31.42**
JSScore	26.09	24.62	27.28	27.71
JSScore + Edit	29.07	28.01	30.57	31.38
Jaccard	27.13	27.56	29.43	29.47
Jaccard + Edit	29.21	**30.38**	30.80	28.85

ceptualization based similarities can improve the search relevance results respectively.

Similarity Measure based on Concepts

Given a pair of URL u and query q, we measure their similarity based on the concept vectors $\mathbf{c}_u$ and $\mathbf{c}_q$. We adopt three similarity scores used in this experiment: (1) cosine similarity, (2) Jaccard similarity, and (3) Jenson-Shannon (JS) divergence. JS divergence is a popular method to measure the similarity between two probability distributions. Although we are not strictly using a probabilistic approach, we use normalized values in the concept vector to approximate a distribution to compute the JS score.

Besides the three similarity scores for concepts, we also test edit distance based similarity, which is used to compare our semantic approach with the string similarity based approach. The edit distance-based similarity is computed based on

$$\text{Edit}(q,u) = \frac{1}{\sum w_{st_i}} \sum_{i=1}^{N} w_{st_i} \left[1 - \frac{\text{EditDist}(q, st_i)}{\max(\text{Len}(q), \text{Len}(st_i))}\right], \qquad (9)$$

where w_{st_i} is defined as the same value as in Eq. (8), EditDist(q, st_i) is the edit distance between two strings of queries q and st_i (the ith short text related to URL), and Len(q) is the length of the string of q.

The results are shown in Tables 7 and 8. Among cosine, Jaccard and JSScore, cosine is the best. Jaccard similarity loses the information of concept weight. Moreover, our approach is not strictly a probabilistic approach. Therefore the JSScore may not be able to best evaluate the similarity between two sets of concepts. Furthermore, we can see that "cosine+edit" gives the best results. We do not argue that concept similarity can solely beat edit distance, since semantically related queries are only a portion of the related queries. There are also a lot of query refinements with similar surface. Therefore, we argue that our signal is a very good complementary signal to simple surface based similarities.

Content + Concepts (Content Ranker)

To show how our method can improve the state-of-the-art methods, we first show how the concept information can be used to improve the content-based ranking scores. We inject our concept based similarity scores to real ranking system, in which the content features include about 1,000 dimensions. Particularly, those features include BM25 and BM25F [34], lexical features, statistical counting features, etc. [7, 23, 35]. The learning to rank machine automatically combines different features to fit the training data [35]. The comparison results are shown in Fig. 6 for mixture data and Fig. 7 for tail data. We can see that the improvement achieved by our approach is larger on tail data than on mixture data. This may be because the popularity fields of mixture data contain more information for each URL. Thus, the additional information provided by Probase is not as significant as it is for the tail data. Moreover, we see that click 10M data and click 5M data achieve more improvements than anchor texts. The historical queries contain more information on both semantics and users' intents.

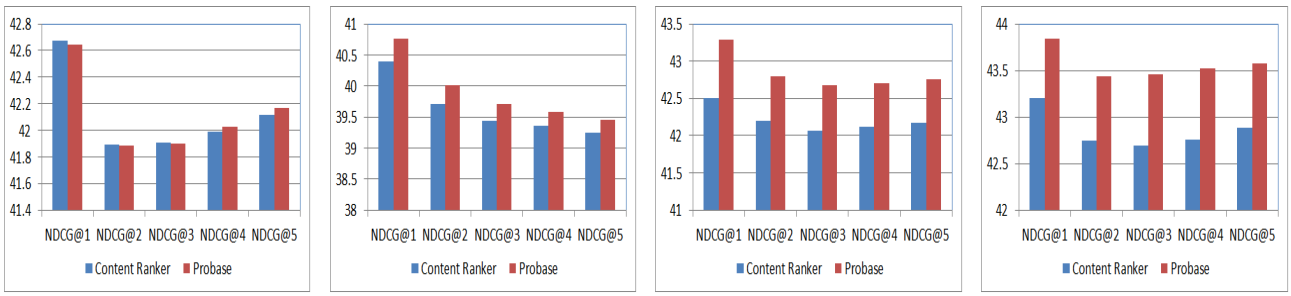
Content + Click + Concepts (Full Ranker)

To check whether the improvement is only achieved through the click information, we also compare with a full ranker. This ranker makes use of both content and click information, including the features generated by click model [38] and translation model [12]. The comparison results are shown in Figs. 8 and 9. We can see that, for the tail data, concept information can also show improvement. However, for the mixture data, only click 10M data shows small improvement. This is because the click information for the mixture data is already good, and the concept does not present enough additional information. The click 10M data shows small improvements while click 5M does not, since click 10M's data contains richer historical queries that can be used to model the intent. Nonetheless, for tail queries, click information is still not enough.

The above experiments show that our method is especially useful when exact click information is not enough to evaluate the relevance score.

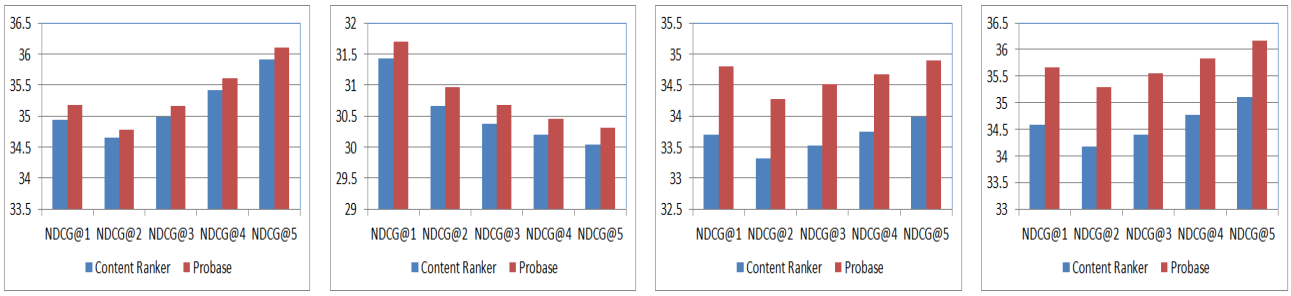
6. CONCLUSION AND FUTURE WORK

In web search, one of the most challenging problems is how to handle tail queries. In this paper, we introduce a novel approach to transfer our understanding of head queries to tail queries. We develop a core technique called conceptualization that maps a short text to concepts defined in a general probabilistic knowledgebase. This allows us to represent queries by concepts with weights. Furthermore, we expand URLs with textual context (by leveraging the click-through graph, anchor texts, web page titles, etc.) so that we can map URLs to concepts as well. Once both queries (head and tail queries alike) and URLs are in the same concept space, we can compute their similarity as a relevance measure. We perform exper-



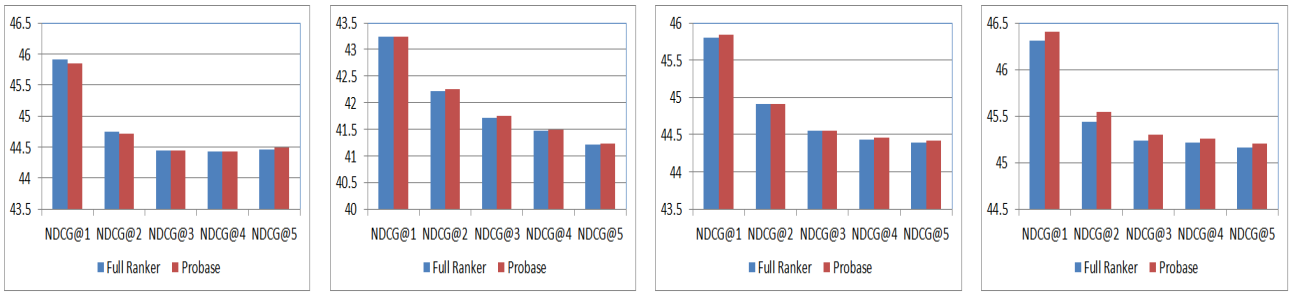
(a) Title. (b) Anchor. (c) Click 5 Months. (d) Click 10 Months.

Figure 6: NDCG results for "mixture" data with content ranker. "Probase" indicates the content ranker with the help of the similarities computed based on Probase.



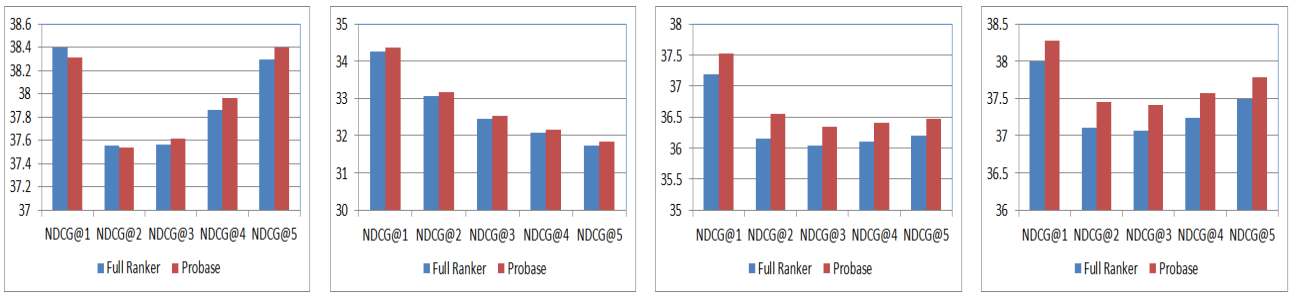
(a) Title. (b) Anchor. (c) Click 5 Months. (d) Click 10 Months.

Figure 7: NDCG results for "tail" data with content ranker. "Probase" indicates the content ranker with the help of the similarities computed based on Probase.



(a) Title. (b) Anchor. (c) Click 5 Months. (d) Click 10 Months.

Figure 8: NDCG results for "mixture" data with full ranker. "Probase" indicates the full ranker with the help of the similarities computed based on Probase.



(a) Title. (b) Anchor. (c) Click 5 Months. (d) Click 10 Months.

Figure 9: NDCG results for "tail" data with full ranker. "Probase" indicates the full ranker with the help of the similarities computed based on Probase.

iments on two real world data sets to show the effectiveness of this approach in improving search relevance, especially for tail queries.

Acknowledgements

We thank Mei Li for her kind help on the evaluation of ranking system. We would like to thank Jun Xu for his helpful discussion. We also thank the reviewers for their valuable comments to improve this paper.

7. REFERENCES

[1] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *IJCAI*, pages 2670–2676, 2007.

[2] A. Berger and J. Lafferty. Information retrieval as statistical translation. In *SIGIR*, pages 222–229, 1999.

[3] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[4] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, pages 1247–1250, 2008.

[5] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. H. Jr., and T. M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, pages 1306–1313, 2010.

[6] K. Collins-Thompson and J. Callan. Query expansion using random walk models. In *CIKM*, pages 704–711, 2005.

[7] P. Donmez, K. M. Svore, and C. J. C. Burges. On the local optimality of lambdarank. In *SIGIR*, pages 460–467, 2009.

[8] D. Downey, S. Dumais, and E. Horvitz. Heads and tails: Studies of web search with common and rare queries. In *SIGIR*, pages 847–848, 2007.

[9] O. Egozi, S. Markovitch, and E. Gabrilovich. Concept-based information retrieval using explicit semantic analysis. *ACM Trans. Inf. Syst.*, 29(2):8:1–8:34, 2011.

[10] O. Etzioni, M. J. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Web-scale information extraction in knowitall: (preliminary results). In *WWW*, pages 100–110, 2004.

[11] C. Fellbaum, editor. *WordNet: an electronic lexical database*. MIT Press, 1998.

[12] J. Gao, X. He, and J.-Y. Nie. Clickthrough-based translation models for web search: from word models to phrase models. In *CIKM*, pages 1139–1148, 2010.

[13] J. Gao, K. Toutanova, and W. tau Yih. Clickthrough-based latent semantic models for web search. In *SIGIR*, pages 675–684, 2011.

[14] M.-H. Hsu and H.-H. Chen. Information retrieval with commonsense knowledge. In *SIGIR*, pages 651–652, 2006.

[15] M.-H. Hsu, M.-F. Tsai, and H.-H. Chen. Combining wordnet and conceptnet for automatic query expansion: a learning approach. In *AIRS*, pages 213–224, 2008.

[16] J. Huang, J. Gao, J. Miao, X. Li, K. Wang, F. Behr, and C. L. Giles. Exploring web scale language models for search query processing. In *WWW*, pages 451–460, 2010.

[17] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck. Learning deep structured semantic models for web search using clickthrough data. In *CIKM*, pages 2333–2338, 2013.

[18] K. Järvelin and J. Kekäläinen. Ir evaluation methods for retrieving highly relevant documents. In *SIGIR*, pages 41–48, 2000.

[19] T. Joachims. Optimizing search engines using clickthrough data. In *KDD*, pages 133–142, 2002.

[20] A. Kotov and C. Zhai. Tapping into knowledge base for concept feedback: Leveraging conceptnet to improve search results for difficult queries. In *WSDM*, 2012.

[21] R. Kraft and J. Zien. Mining anchor text for query refinement. In *WWW*, pages 666–674, 2004.

[22] D. B. Lenat and R. V. Guha. *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*. Addison-Wesley, 1989.

[23] P. Li, C. J. C. Burges, and Q. Wu. Mcrank: Learning to rank using multiple classification and gradient boosting. In *NIPS*, 2007.

[24] Y. Li, W. P. R. Luk, K. S. E. Ho, and F. L. K. Chung. Improving weak ad-hoc queries using wikipedia as external corpus. In *SIGIR*, pages 797–798, 2007.

[25] S. Liu, F. Liu, C. Yu, and W. Meng. An effective approach to document retrieval via utilizing wordnet and recognizing phrases. In *SIGIR*, pages 266–272, 2004.

[26] R. Mandala, T. Tokunaga, and H. Tanaka. Combining multiple evidence from different types of thesaurus for query expansion. In *SIGIR*, pages 191–197, 1999.

[27] S. P. Ponzetto and M. Strube. Deriving a large-scale taxonomy from wikipedia. In *AAAI*, pages 1440–1445, 2007.

[28] M. Richardson, E. Dominowska, and R. Ragno. Predicting clicks: estimating the click-through rate for new ads. In *WWW*, pages 521–530, 2007.

[29] S. Robertson, H. Zaragoza, and M. Taylor. Simple bm25 extension to multiple weighted fields. In *CIKM*, pages 42–49, 2004.

[30] E. Sadikov, J. Madhavan, L. Wang, and A. Halevy. Clustering query refinements by user intent. In *WWW*, pages 841–850, 2010.

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

[32] Y. Song, H. Wang, Z. Wang, H. Li, and W. Chen. Short text conceptualization using a probabilistic knowledgebase. In *IJCAI*, pages 2330–2336, 2011.

[33] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *WWW*, pages 697–706, 2007.

[34] K. M. Svore and C. J. C. Burges. A machine learning approach for improved bm25 retrieval. In *CIKM*, pages 1811–1814, 2009.

[35] Q. Wu, C. J. C. Burges, K. M. Svore, and J. Gao. Adapting boosting for information retrieval measures. *Inf. Retr.*, 13(3):254–270, 2010.

[36] W. Wu, H. Li, H. Wang, and K. Q. Zhu. Probase: A probabilistic taxonomy for text understanding. In *SIGMOD*, pages 481–492, 2012.

[37] Z. Yin, M. Shokouhi, and N. Craswell. Query expansion using external evidence. In *ECIR*, pages 362–374, 2009.

[38] Y. Zhang, W. Chen, D. Wang, and Q. Yang. User-click modeling for understanding and predicting search-behavior. In *KDD*, pages 1388–1396, 2011.