Toward Topic Search on the Web

ABSTRACT

Traditional web search engines treat queries as sequences of keywords and return web pages that contain those keywords as results. Such a mechanism is effective when the user knows exactly the right words that web pages use to describe the content they are looking for. However, it is less than satisfactory or even downright hopeless if the user asks for a concept or topic that has broader and sometimes ambiguous meanings. This is because keyword-based search engines index web pages by keywords and not by concepts or topics. In fact they do not *understand* the content of the web pages. In this paper, we present a framework that improves web search experiences through the use of a probabilistic knowledge base. The framework classifies web queries into different patterns according to the concepts and entities in addition to keywords contained in these queries. Then it produces answers by interpreting the queries with the help of the knowledge base. Our preliminary results showed that the new framework is capable of answering various types of topic-like queries with much higher user satisfaction, and is therefore a valuable addition to the traditional web search.

1. INTRODUCTION

Keyword based search works well if the users know exactly what they want and formulate queries with the "right" keywords. It does not help much and is sometimes even hopeless if the users only have vague concepts about what they are asking. The followings are four examples of such "conceptual queries":

Q1. database conferences in asian cities
Q2. big financial companies campaign donation
Q3. tech companies slogan
Q4. winter vacation destinations except florida

Although the intentions of these queries are quite clear, they are not "good" keyword queries by traditional standard. In the first query, the user wants to know about the database conferences located in Asian cities, without knowing the names of the conferences or cities. In the second query, the user asks which big financial companies are involved in campaign donation. In the third query, the user wants to find out the various slogans of tech companies. In the fourth query, the user tries to get information about winter vacation places excluding Florida.

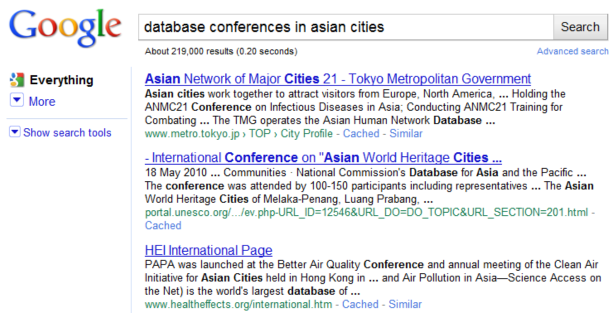


Figure 1: Google results for Q1

Figure 2: Google results for Q2

Fig. 1 through 4 show the top results returned by Google and Bing for these four queries. None of these results render meaningful answers to the actual questions implied by the queries. Most of them are likely to be considered irrelevant by the user. These results were returned merely because they contain some of the keywords in the queries. For example, the top two results in Fig. 1 have nothing to do with Asian cities, or database conferences. In Fig. 2, the words "financial" and "finance" appear in various places but are not related to financial companies. Fig. 3 happens to return a page with a list of company slogans, not because these are tech companies but because one of the company names contains the word "tech". Fig. 4 shows that Bing does not understand the meaning of "except florida" and returns pages that are about vacations *in* Florida.

Apparently, *database conferences*, *asian cities*, *big financial companies* and *winter vacation destinations*, *etc.* are abstract concepts while *slogan* can be regarded as an attribute of tech companies. None of these are keywords in the traditional sense. Traditional search engines are good at handling entities in these classes (e.g.

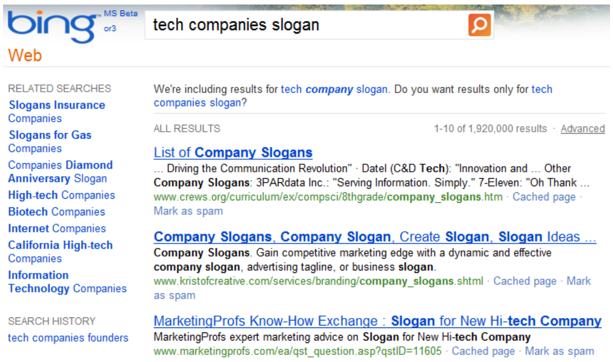


Figure 3: Bing results for Q3

Figure 4: Bing results for Q4

VLDB as an entity of database conferences), because these entities can be exactly matched as keywords. But in reality, quite a significant percentage of web queries are not *entity only* queries. Our statistics from the search log of a major search engine in last two years suggests that about 62% of the queries contain at least one conceptual class term (see Fig. 11). To better serve such conceptual queries, we need to understand concepts in web pages.

In this paper, we present a framework that leverages a knowledge base and query interpretation techniques to improve web search on concept-related web queries. This knowledge base, named Probase[14], is a general-purpose probabilistic taxonomy, automatically constructed by integrating information from web pages and other more reliable data sources such as Freebase [11] and Wordnet [16]. It contains large number of concepts, entities and attributes, organized in a hierarchical structure with subsumption, similarity and other relations. Here, an *entity* refers to a specific object, a *concept* refers to a collection of things, and an *attribute* refers to a property of one or more objects. With the help of Probase, we can identify concepts and attributes in queries and interpret them by replacing the concepts with their most likely entities, and hence formulate more accurate keyword-based queries. The results of these new queries from the traditional search engines can then be ranked and presented to users, in addition to normal keyword queries.

Fig. 5 shows the top search results of the query "database conferences in asian cities" from our prototype system. These results do not contain the keywords "database conferences" or "asian cities", but instead directly gives information about three recent VLDB conferences that were actually hosted in Asia. This information is a lot more targeted and relevant from the user perspective. Fig. 6 gives the top results for query "big financial companies campaign donation" on our system. Our system correctly links Citigroup and

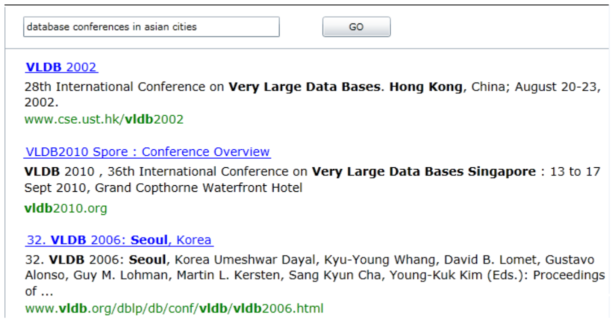


Figure 5: Our top results for Q1

Figure 6: Our top results for Q2

Merrill Lynch, two of the largest financial institutes with contributions to political campaigns. When serving the query "tech companies slogan", our prototype system realizes that the information is immediately available from the taxonomy, and hence presents all the slogans it knows directly in a table, which is shown in Fig. 7. Fig. 8 pinpoints accurately some of the top winter vacation destinations other than Florida, such as Mexico and Hawaii.

It is important to note that the lack of a concept-based search feature in all mainstream search engines has, in many situations, discouraged people from expressing their queries in a more natural way. Instead, users are forced to formulate their queries as keywords. This makes it difficult for people who are new to keyword-based search to effectively acquire information from the web.

The proposed framework is not meant to replace the existing keyword based search, but complements keyword search as we can now handle some of "non-keyword" queries as well. This framework therefore represents a significant step toward a new genre of search on the web — *topic search.*

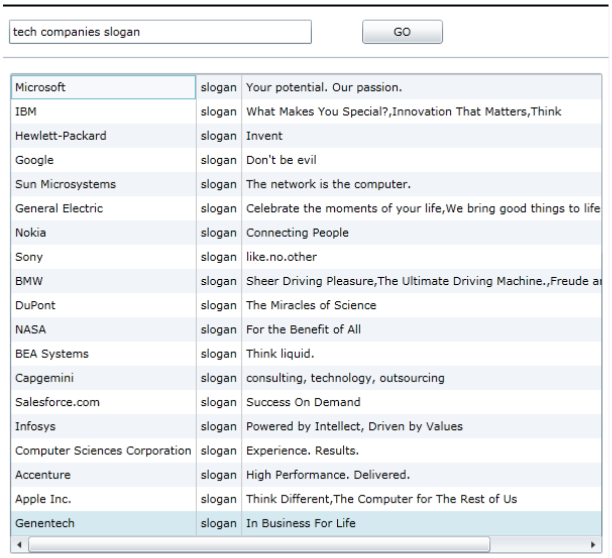
The main contributions of this paper are:

1. a large-scale, automatically constructed taxonomy is used for web search;
2. the new framework better understands user queries by pattern matching and query interpretation using the concepts and entities in the taxonomy;
3. our experiments show that this conceptual search framework provides additional values for topic-related queries and significantly improves user experiences with web search;

In the remainder of this paper, we will first give an overview of the topic search framework (Section 2), followed by the introduction of Probase (Section 3). Section 4 details the proposed framework, and Section 5 presents the evaluation of the system. This is followed by some most related work (Section 6) and the conclusion (Section 7).

2. OVERVIEW

In web search, limited query rewriting techniques are used to find a term's alternative forms, so that documents containing the



Microsoft	slogan	Your potential. Our passion.
IBM	slogan	What Makes You Special?,Innovation That Matters,Think
Hewlett-Packard	slogan	Invent
Google	slogan	Don't be evil
Sun Microsystems	slogan	The network is the computer.
General Electric	slogan	Celebrate the moments of your life,We bring good things to life
Nokia	slogan	Connecting People
Sony	slogan	like.no.other
BMW	slogan	Sheer Driving Pleasure,The Ultimate Driving Machine.,Freude ai
DuPont	slogan	The Miracles of Science
NASA	slogan	For the Benefit of All
BEA Systems	slogan	Think liquid.
Capgemini	slogan	consulting, technology, outsourcing
Salesforce.com	slogan	Success On Demand
Infosys	slogan	Powered by Intellect, Driven by Values
Computer Sciences Corporation	slogan	Experience. Results.
Accenture	slogan	High Performance. Delivered.
Apple Inc.	slogan	Think Different,The Computer for The Rest of Us
Genentech	slogan	In Business For Life

Figure 7: Our top results for Q3

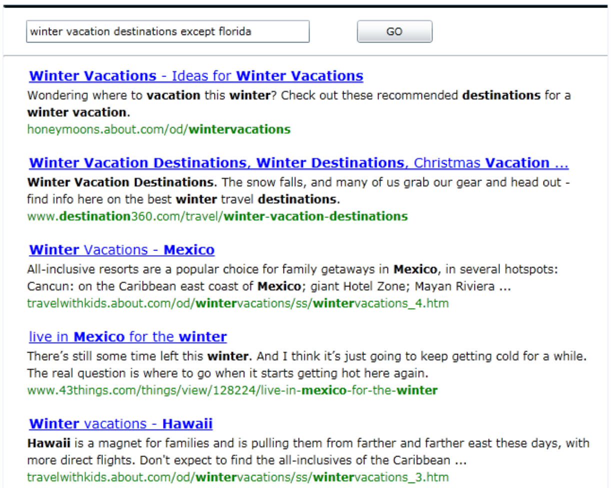


Figure 8: Our top results for Q4

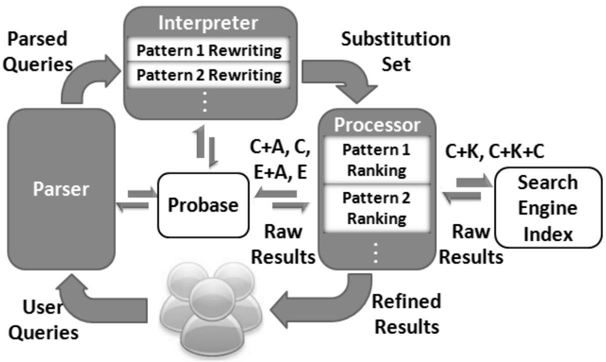


Figure 9: The framework of topic search

alternative forms are also considered. In this paper, we take query rewriting to a new level by using a large-scale probabilistic taxonomy named "Probase" for understanding. This enables us to better interpret user queries. Fig. 9 shows the framework of our system, which is centered around the taxonomy.

More specifically, our framework is comprised of three main modules: the *parser*, the *interpreter* and the *processor*. When a user issues a query, the *parser* uses Probase to decompose the query into possible term sequences, which consist of terms of 4 different types: concepts, entities, attributes and keywords. The *interpreter* identifies the intent or the semantics of the term sequence based on a set of query patterns. In this paper, we focus on five most useful patterns and their combinations. The interpreter rewrites the parsed queries into a set of candidate queries by substituting abstract concepts with their specific entities. The *processor* ranks the candidate queries based on their likelihood, which is estimated by word association probabilities. The processor then submits top queries either to Probase or to the search engine index to obtain a list of raw results. It ranks the results in a way similar to a normal search engine before returning the final results to the user.

3. THE KNOWLEDGE BASE

In order to better understand queries, the search engine needs to have access to a knowledge base, which knows that, for example, *VLDB* is a database conference, *Hong Kong* is an Asian city, *except Florida* means the other 49 states in the US, many companies have their *slogans*, and the slogan of Google, a well known tech companies, is "Don't be evil." Furthermore, we also need certain meta information, for example, how entities are ranked by their representativeness within a same concept (e.g., What are the top 5 Internet companies?), or how plausible is a claim (e.g., Is Pluto a planet, or a dwarf planet?)

Unfortunately, few existing knowledge bases are qualified for this purpose. This is because, first, although there exists a large number of ontologies and taxonomies, they are often domain specific, and therefore very hard to integrate. Second, universal taxonomies, including the well known Cyc [25] and Freebase [11], have a relatively small conceptual scope: Cyc has about 120 thousand concepts while Freebase contains only about 1,500 concepts in shallow conceptual structures. This contrasts with the rich conceptual structure in a human mind.

Figure 10: A fragment of Probase taxonomy

In this work, we take advantage of the Probase taxonomy for rewriting search queries. The backbone of Probase is constructed using linguistic patterns such as Hearst patterns [20]. For example, a sentence that contains "... politicians such as Barack Obama and Tony Blair..." can be considered as an evidence for the claim that *politicians* is a hypernym of *Barack Obama* and *Tony Blair*. Fig. 10 illustrates a snapshot of the Probase taxonomy which includes the concept “politicians”, as well as its super-concepts, sub-concepts, entities and similar concepts.

Probase is unique in two aspects. First, the Probase taxonomy is very *rich*. The core taxonomy alone (which is learned from 1.68 billion web pages and 2 years’ worth of search log from a major search engine) contains around 2.7 million concepts. The rich set of super-concepts, sub-concepts, and similar concepts of *politicians* shown in Fig. 10 is an example. Indeed, with 2.7 million concepts obtained directly from Web documents, the knowledge base has much better chance to encompass as many concepts in the mind of humans beings as possible. As shown in Fig. 11, at least 80% of the search contains concepts or entities that appear in Probase.

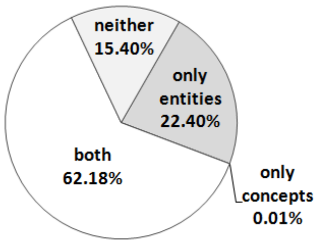


Figure 11: Percentage of searches with concepts or entities

Second, the Probase taxonomy is *probabilistic*, which means every claim in Probase is associated with some probabilities that model the claim’s plausibility, ambiguity, and other characteristics. The probabilities are derived from evidences found in web data, search logs, and other available data. Because of the probabilistic framework, it is natural for Probase to integrate information from other data sources, including other ontologies. It also enables Probase to rank the information it contains. For example, it can answer questions such as “What is the top 5 Internet companies?”, or “How likely is Pluto a planet vs. a dwarf planet?”

4. OUR APPROACH

Our topic search framework contains the following modules.

4.1 Query Parsing

We regard a search query as a sequence of *terms*. We are interested in five kinds of terms: *entity* terms, *concept* terms, *attribute* terms, *keyword* terms and *auxiliary modifiers*. Keywords are non-trivial types of words other than concepts and entities, and auxiliary modifiers modify a class of things. Table 1 gives some examples of each type of terms which are highlighted.

The first three types are usually noun phrases, while the keyword terms are often verbs, adjectives or any combinations of other terms that are not recognized as one of the first three types. The last type is special patterns consisting of an auxiliary term (such as “besides”, “except”, “including”, etc.) plus one or more noun phrases.

Formally, we represent a raw query by an array of words, that is, $q[1,n] = (w_1, \cdots, w_n)$. We parse it into a sequence of terms, where each term is an entity, a concept, an attribute, or an attribute value in the Probase taxonomy, or simply a keyword otherwise. Specifically, we represent a parsed query as $p[1,m] = (t_1, \cdots, t_m)$, where each t_k is a consecutive list of words in the raw query, i.e., $t_k = q[i,j] = (w_i, w_{i+1}, \cdots, w_j)$.

Clearly, there may exist many possible interpretations of a query, or multiple different parses. For example: query “*president george bush fires general batiste*” can be parsed as

[president] (george bush) fires [general] (batiste)

Term Type	Examples
Entity	**Citigroup** **ICDE** in **Hong Kong** **Hong Kong** area
Concept	**companies** **big financial companies** campaign donation **database conferences** in **asian cities**
Attribute	tech companies **slogan** Hong Kong **area** movies **director**
Keyword	Oracle **acquire** Sun big financial companies **campaign donation** **what’s the date today**
Auxiliary Modifiers	winter vacation destinations **except Florida** IT companies **besides Microsoft and Google** asian cities **other than Singapore and Hong Kong**

Table 1: Query Terms and Examples

[president] george [bush fires] [general] (batiste)

(president) (george bush) fires [general] (batiste)

where () denotes an entity, [] a concept, <> an attribute. The reason of multiple parses is because both *george bush* and *bush fires* are valid terms in Probase. Further more, *president* can either be a concept, which refers to all presidents, or a specific entity in the political leader concept. The parser needs to return all meaningful parses from a query.

For an n-word query, there are $2^{(n-1)}$ possible parses which is expensive to compute. We first introduce a greedy algorithm to solve this problem. Then we improve this algorithm using a dynamic programming approach. The greedy algorithm contains three steps: (1) find all possible terms; (2) find all correlations among terms; (3) use a scoring function to find one meaningful parse in a greedy manner.

First, we find all terms in a query. For a sequence of n word, there are $n(n+1)/2$ possible subsequences. We check each of them to see if they are concepts, entities or attributes. We give a term t a *score* according to its type and length:

$$s_{term}(t) = w_{term}(t) \cdot w_{len}(|t|) \tag{1}$$

where $|t|$ is the number of words in t, $w_{term}(t)$ is the weight function defined as:

$$w_{term}(t) = \begin{cases} w_e, & \text{if t is an entity} \\ w_c, & \text{if t is a concept} \\ w_a, & \text{if t is an attribute} \\ 0, & \text{otherwise} \end{cases}$$

and

$$w_{len}(x) = x^{\alpha}$$

where w_e, w_c and w_a are constants, and $w_e > w_c$. We let $\alpha > 1$ to bias toward longer terms.

Next, we consider the correlations among terms. Currently we focus on three kinds of correlations: Entity-Attribute, Concept-Attribute, and Concept-Entity. We use R_1-R_2 to denote the correlation between one R_1 term and several R_2 terms. For example, “<population> of (china)” is an instance of Entity-Attribute correlation, and “[tech companies] <slogan> and <founder>” is an instance of Concept-Attribute correlation. Note that terms in a correlation do not have to be physically adjacent to each other, which means keywords can be mixed with correlated terms, e.g. "[presidents] and their <wives>".

Based on terms and term scores, we define *block* and *block score*. A block is either a correlation or a single term. The block score is defined as:

$$s_{block}(q[i,j]) = \max_{i',j'}\{w_{block}(p[i',j']) \cdot \sum_{k=i'}^{j'} s_{term}(t_k)\} \quad (2)$$

where $p[i',\ j']$ is a term sequence parsed from $q[i,\ j]$, and

$$w_{block}(p[i',j']) = \begin{cases} w_{e-a}, & \text{if p[i',j'] is an E-A correlation} \\ w_{c-a}, & \text{if p[i',j'] is a C-A correlation} \\ w_{c-e}, & \text{if p[i',j'] is a C-E correlation} \\ 1, & \text{otherwise} \end{cases}$$

where w_{e-a}, w_{c-a} and w_{c-e} are all greater than 1. The above formula rewards blocks with a term correlation, and if there is no correlation, the block score is equal to the sum of term scores.

Finally, after finding all possible terms and blocks, we greedily select the block with the highest score. Once a block is selected, all blocks it overlaps with are removed.

We show the three-step greedy algorithm for parsing query "*president george bush fires general batiste*" in Fig. 12. In step (1), we identify all terms in the query and score them according to Eq. 1. In step (2), we generate blocks based on these terms. Each term becomes a block and their block scores equal to their term scores. At the same time, we notice that (george bush) is an entity of concept [president], so we build a C-E correlation block. Same goes for (batiste) and [general]. In step (3), we perform greedy selection on blocks. We identify "[president] (george bush)" as the best block among all, and remove other overlapping blocks such as "[president]", "(george bush)" and "[bush fires]". Similarly we keep block "[general] (batiste)" and remove its overlapping blocks. As a result, the algorithm returns "[president] (george bush) fires [general] (batiste)" as the best parse.

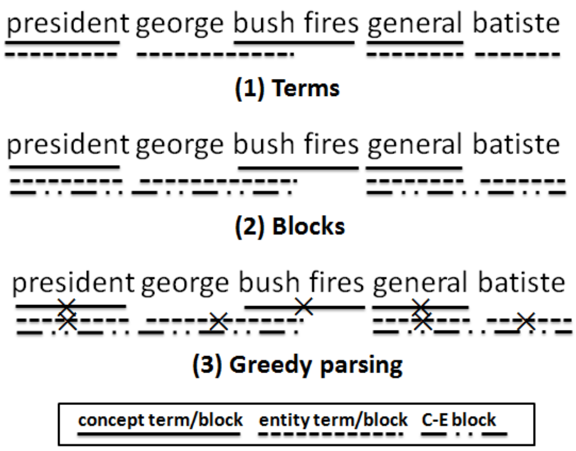


Figure 12: Example of greedy parsing

However, the greedy algorithm does not guarantee an optimal parse, and it cannot return a list of top parses. As an improvement, we propose the following dynamic programming algorithm. We define a *preference score* $s_{pref}(n)$ to represent the quality of the best parse of a query of n words:

$$s_{pref}(n) = \begin{cases} \max_{i=0}^{n-1}\{s_{pref}(i) + s_{block}(q[i+1,n])\}, & \text{if n} > 0 \\ 0, & \text{if n} = 0 \end{cases}$$

By memorizing the sub-solutions of $s_{pref}(n)$, one can produce the parse with highest preference score. Moreover, when defining $s_{pref}(n)$ as a score set of top parses, one can also obtain the top k parses.

4.2 Query Interpretation

In this module, we classify the input parsed queries into different patterns. Our analysis on the search log during the period of September of 2007 to June of 2009 of a major search engine (see Fig. 11) shows that about 62% of the queries contain at least one concept term. More detailed analysis revealed that common web queries can be classified into a number of different patterns. The following six basic patterns account for the majority of all the queries during that period:

1. *Single Entity (E)*
2. *Single Concept (C)*
3. *Single Entity + Attributes (E+A)*
4. *Single Concept + Attributes (C+A)*
5. *Single Concept + Keywords (C+K)*
6. *Concept + Keywords + Concept (C+K+C)*

These patterns can be combined to form more complex patterns. Table 2 lists some example queries of each pattern.

Table 2: Query Patterns and Examples

Patterns	Queries
E	(VLDB) (Citigroup)
C	[database conferences] [big financial companies]
E+A	(Apple) <slogan> [Hong Kong] <country> <area>
C+A	[tech companies] <slogan> [films] <language> <tagline>
C+K	[big financial companies] campaign donation [species] endangered
C+K+C	[database conferences] in [asian cities] [politicians] commit [crimes]

Once the system determines the pattern of each parsed query, it starts interpreting them using the following strategies. The general approach is substituting the abstract concepts in a query with more specific search terms such as their associated entities which are more suitable for traditional keyword search.

For *E* and *E+A* queries, no further interpretation is necessary since this type of queries are already specific and can be searched directly in both Probase and the search engine index.

For a *C* or *C+A* query, it substitutes a list of top entities associated with that concept in Probase for the concept term to form a list of E or E+A queries.

A special case is when the concept is *implicitly* implied by an auxiliary modifier such as "... *except florida*". Here, "florida" is an entity in Probase, and we treat [*except florida*] as if it is a concept. And this concept is the most *representative* concept in which "florida" is an entity but with "florida" removed from it. To find the most representative concept to an entity, we use a DF-ITF score [1] which is a form of *inverse* function of the well-known TF-IDF score [32].

For a *C+K* query, it replaces the concept with its associated entities to form a list of Entity + Keywords queries which require no further interpretation.

For a *C+K+C* query, it is considered as a extended form of C+K queries. We replace both concepts with their associated entities to form a list of Entity + Keywords + Entity queries. Note that the number of such queries can be very large but we will show in the next subsection how to reduce them to only relevant queries.

Finally, the output of the Interpreter module is a set of substituted queries of the following 4 patterns: E, E+A, E+K and E+K+E.

4.3 Query Processing

The Processor module takes as input a set of substituted candidate queries, submits some or all of these queries to Probase or a search index, and presents a final set of ranked results to the user.

For E and E+A pattern queries, the processor queries the Probase taxonomy for all the detailed information about this particular entity. This information is returned as a table which will eventually be presented to the user as an info-box (e.g. Fig. 7).

In the rest of this subsection, we will focus on E+K and E+K+E queries which are more interesting and challenging to process. One naive approach is to submit all these substituted queries to a traditional search engine index, combine the results, and present ranked results to the user. However, number of such queries can be prohibitively large because many concepts are associated with large number of the entities. For example, Probase contains hundreds of *politicians* and thousands of *crimes*. For query "*politicians commit crimes*", the system would generate millions of candidate substitutions even though most of these, such as "*obama commit burglary*", are not relevant at all.

The key technical challenge is understanding user's intent and filtering out those substituted queries which are not relevant. We observe that in most queries, keywords act as modifiers that limit the scope of the accompanying concept in C+K queries, or imposing a particular relationship between the two surrounding concepts in C+K+C queries. For example by specifying "commit" in the above example, it excludes other relationships such as "politicians *fight* crimes" or "politicians *victimized by* crimes".

Our proposed technique to address the above problem is compute the *word association* values between an entity and a sequence of keywords, and multiply that with the representativeness score of this entity in the concept it belongs to, to get a *relevance score* for a given query.

4.3.1 Word Association

A word association value is used to measure the associativity of a set of words. For example, word *Microsoft* and word *technology* have a high association value than *Walmart* and *technology*, because Microsoft is a technology company whereas Walmart is a supermarket presumably less "technology-savvy". To compute the word association value between two words, which we call *two-way association*, we measure the frequency of *co-occurrence* of the two words in a document among all documents or in a sentence among all sentences in the all documents. The concept can be easily extended to association of more than two words, namely *multi-way association*.

Word association values among a fixed set of words are often precomputed and stored in a matrix for efficient runtime lookup. Computing a two-way association matrix for a set of N words generally takes $O(N^2)$ time, while computing a multi-way matrix is even more costly. Li [27] proposed a method to estimate multi-way association. It uses an improved sketch method when sampling inverted index list of the words and then use maximum likelihood estimation to solve the problem. Consequently, one only needs to store the sampled inverted list of each word, instead of all the association values of different word combinations. However, scanning the inverted list remains to be costly if the number of documents is too large, especially at the scale of the entire web.

In this paper, we approximate multi-way association by combining the values of two-way association. In general, given a set of m words: $W = \{w_1, w_2, ..., w_m\}$, it is impossible to compute the exact word association $wa(W)$ based only on two-way association $wa(\{w_i,\ w_j\})$, where w_i and w_j are any two distinct words in the W. This is because the co-occurrence of all words in W may be independent of the co-occurrence of any pair of two words in W. However, because an co-occurrence of W together implies a co-occurrence of (w_i, w_j), for any $w_i, w_j \in W$, we have

$$wa(W) \leq \min\ wa(\{w_i, w_j\}).$$

In other words, the minimum two-way association value provides an upper bound to the m-way association value. In this paper, as we will show later in the section, we are actually computing a special m-way association involving the words in an entity term and the words in a keyword term. In this computation, we approximate the m-way association, by the minimum value of the two-way association between a word in the entity and the key word, or

$$wa(\{e, w_{key}\}) \approx \min_{w_i \in e} wa(\{w_i, w_{key}\}). \tag{3}$$

This technique is based on the notion of *pivot words*. A pivot word is the most informative and distinguishing word in a short phrase or term. This word is so special to the term, that it allows people to recognize the term even without looking at the other words. Given that W contains the words from an entity term and a keyword, if two words w_i and w_j from W co-occur the minimum number of times, we argue that there is a high probability that one of them is the pivot word of the entity term and the other is the keyword. This is because the pivot word appears less frequently than the other more common words in the entity. It is even rarer for the pivot word to appear with an arbitrary keyword than with the other words in the entity. Therefore $wa(\{e_{pivot}, w_{key}\})$ is likely to be minimum. On the other hand, because the pivot word is special and distinguishing, when it does appear in the text, it often appears in the context of the whole entity term, and therefore $wa(\{e_{pivot}, w_{key}\})$ can be used to simulate $wa(\{e, w_{key}\})$.

We can further extend (3) to compute the association of an entity term and a sequence of keywords:

$$wa^{+}(\{e, k_1, \ldots k_n\}) \approx \min_{i \in [1,n]} wa(\{e, k_i\}). \tag{4}$$

To get the word association values, we first obtain a list of most frequently used words on the web (excluding common stop words) as our word list and then compute the sentence-level pairwise co-occurrence of these words. We chose to count the co-occurrence within sentences rather than documents because this gives stronger evidence of association between two words.

4.3.2 E+K and E+K+E queries

We first group the E+K and E+K+E queries by their prefix E+K. As a result, each E+K query form a group by itself; and E+K+E queries with the same prefix form a group. Next, we compute a relevance score for each group G as

$$\max_{q \in G}(wa^{+}(q_{e1}, q_k, q_{e2}) \times rp(q))$$

where q_{e1}, q_k and q_{e2} are the first entity, keywords and the second entity of query q (q_{e2} may be null if q is an E+K query), and $rp(q)$ is the representativeness score of q (see Algorithm 1).

We then select the top n groups with the best relevance scores. For the n groups, if a group G is an E+K group, we simply send the only query contained in this group to the search index and gather the results as the final results; if a group G is an E+K+E group, we use a two-pass procedure that accesses the search engine twice: in the first pass, we query the search engine with $\{q_{e1}, q_k\}$, *i.e.* the prefix of the group. Let us call the set of top pages thus returned R. We then remove from G all queries whose second entity, *i.e.* q_{e2}, does not appear in any of the pages in R. In the second pass, we send the remaining queries in G to search engine. Finally we collect all results from the top n groups and rank them according to some common search engine metric such as PageRank before returning to the user. The complete algorithm is listed in Algorithm 1.

Algorithm 1 Processing E+K and E+K+E Query Groups

Input: a set of E+K and E+K+E queries S; argument n.
Output: final search result set $R(S)$.

$S_g \Leftarrow$ group S by sequence prefix {E, K}
for all query group $G \in S_g$ **do**
  for all query $q \in G$ **do**
    $wa(q) \Leftarrow wa^+(q_{e1}, q_k, q_{e2})$
    $rp(q) \Leftarrow$ representativeness score of $e1$ in Probase
    $score(q) \Leftarrow wa(q) \times rp(q)$
  end for
  $score(G) \Leftarrow \max_{q \in G}\{score(q)\}$
end for
$S_{top} \Leftarrow$ top n groups in S_g
$R(S) \Leftarrow \emptyset$
for all query group $G \in S_{top}$ **do**
  if G is an E+K query group **then**
    send the only query $q \in G$ to search engine
    $R(G) \Leftarrow$ top returned results
  else
    $G_{filtered} \Leftarrow \emptyset$
    send query $q_{prefix} = \{q_{e1}, q_k\}$ to search engine
    for all result $r \in R(\{q_{prefix}\})$ **do**
      for all query $q = \{q_{e1}, q_k, q_{e2}\} \in G$ **do**
        if sequence q appears in r **then**
          $G_{filtered} \Leftarrow G_{filtered} \cup \{q\}$
        end if
      end for
    end for
    $R(G) \Leftarrow \emptyset$
    for all query $q \in G_{filtered}$ **do**
      send q to search engine
      $R(\{q\}) \Leftarrow$ returned results
      $R(G) \Leftarrow R(G) \cup R(\{q\})$
    end for
  end if
  $R(S) \Leftarrow R(S) \cup R(G)$
end for

4.4 Ranking Results

Even though the Processor module filters out most irrelevant candidate queries, the number of relevant queries can still be large. The current framework requires all results be returned from the search index before ranking and presenting the final results to the user, which can be time consuming. We therefore made a few optimizations for the C+K and C+K+C queries, two of the most expensive query patterns.

To do this, we set up an empty pool, and send the queries in the top n groups one by one to the search index and place results into the pool. At the same time, we pop the best result in the pool and return it to the user at regular intervals. We continue this process until sufficient results are shown to the user. Algorithm 2 describes it in detail.

Algorithm 2 Ranking

Input: a grouped and ranked query set S_g; the number of top results n.
Output: a list of results $R(S_g)$.

$pool = \emptyset$
for $i \in 1..n$ **do**
  get results $R(G_i)$ of the ith query group G_i in S_g
  $pool \Leftarrow pool \cup R(G_i)$
  pop and show the best result r in $pool$
end for

5. EVALUATION

In this section, we evaluate the performance of online query processing and offline precomputation (word association). To facilitate this evaluation, we create a set of benchmark queries that contain concepts, entities, and attributes, for example, "politicians commit crimes" (C+K+C), "large companies in chicago" (C+K), "president washington quotes" (E+A), etc. For a complete list of the queries and their results, please see [1].

5.1 Semantic Query Processing

Given a query, we analyze the concepts, entities, and attributes in the query. For C+K and C+K+C queries, we rewrite the query by replacing the concepts with appropriate entities. We then send the rewritten queries to Bing using Microsoft Bing API. We then combine their results and compare them with the result of the original query. For other types of queries (e.g., E, C, E+A and C+A), we search for relevant information in Probase, and return a table that contains entities, their attributes and values.

The computation is done on a workstation with a 2.53 GHz Intel Xeon E5540 processor and 32 GB of memory running 64-bit Microsoft Windows Servers 2003. In all search quality related experiments, we asked three human judges to rate each search result as being "relevant" or "not relevant", and we record the majority vote.

Quality of C+K & C+K+C queries

In the first experiment, we compare the user ratings of the top 10 search results returned by our prototype system, Bing and Google, which is illustrated in Fig. 13. It shows that our prototype has a clear advantage at answering concept related queries. In addition, it also shows that results for C+K+C queries have lower relevance than C+K queries across the three systems, because C+K+C queries often involve more complicated semantic relations than C+K queries.

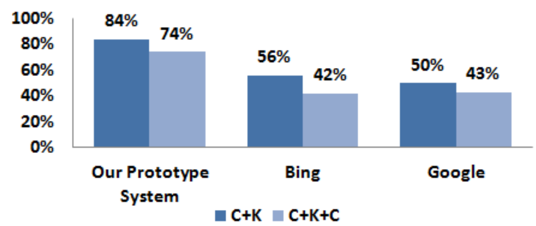
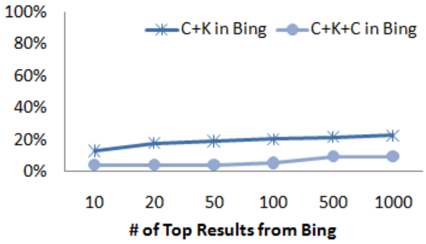
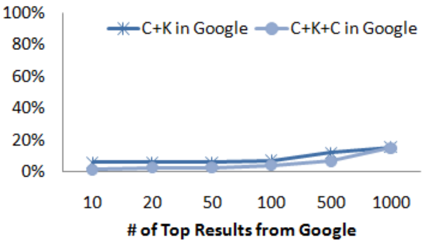


Figure 13: % of relevant results for C+K & C+K+C queries

In the second experiment, we show the ability of our prototype system to pick up relevant results that keyword based search engines would miss. To see this, we focus on relevant results in the top 10 results returned by our prototype system. Among the top 10 results for the 10 C+K queries we use, 84 are judged relevant. The number for C+K+C queries is 74. We check whether these results will ever show up in Google or Bing. Fig. 14 shows that at least 77% of the relevant C+K results and 85% of the relevant C+K+C results in our top 10 could not be found in Bing or Google even after scanning the first 1000 results.



(a) Bing



(b) Google

Figure 14: Bing/Google miss the relevant results in our top 10

In the third experiment, we compare the percentage of relevant results from our prototype system and other semantic search engines including Hakia, Evri and SenseBot like the first experiment. Fig. 15 shows that our system outperforms the three. Besides, they have even fewer relevant results than Google and Bing do. We believe one reason is that they index limited number of documents. For example, we found SenseBot's top 10 results are often extracted from 3∼5 documents.

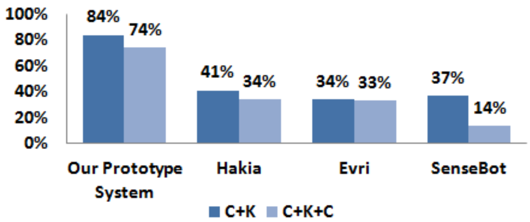


Figure 15: % of relevant results for C+K & C+K+C queries

Quality of E, C, E+A & C+A queries

For E, C, E+A, and C+A queries (see [1]), we return tables containing relevant entities instead of hyperlinks. We ask human judges to rate the relevancy of each returned table. Fig. 16 shows the percentages of relevant results.

Fig. 16 shows that all returned results are relevant for E and E+A queries. The system returned some wrong answers for C and C+A queries, Probase contains some erroneous concept-entity pairs. For example, at present, it is cannot disambiguate "George Washington" as a book or a person.

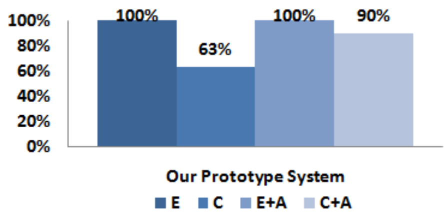


Figure 16: % of relevant results for E, C, E+A, C+A queries

Time Performance

We next evaluate the efficiency of our prototype system. Table. 3 shows the average running time of the benchmark queries in different categories. Note that the system is configured to return only the top 10 results for C+K, C+K+C, C and E patterns, and all results for the other two patterns.

Pattern	Pr.	It.	Pc.	First Result	Total
E	0.06	0.16	0.16	0.38	0.38
C	0.33	0.23	0.32	0.88	0.88
E + A	0.15	0.16	0.08	0.39	0.39
C + A	0.16	0.66	0.15	0.97	0.97
C + K	0.12	0.50	5.24	0.62	5.87
C + K + C	0.36	1.22	13.21	2.83	14.79

Table 3: Execution Time (secs)

* Pr. = Parsing, It. = Interpretation, Pc. = Processing.

We currently use only one machine to communicate with Bing's public API to support our system. The API accepts only 2 queries per second. So we can see that C+K & C+K+C queries take much more time on processing than other queries. In this paper, we focus on improving user experience by presenting the first result as soon as it becomes available instead of showing all results at the end. [1] We can also find that C+K+C queries take more time to process than C+K ones because two round trips to Bing are required for each query – one for filtering and one for final results. On the other hand, C queries take more time than E since it contains multiple Es. C+A queries require more time than C queries because we need to remove entities without any attributes in the query. For the same reason, E+A queries take longer than E.

We present the traffic statistics for C+K and C+K+C queries, the only two types of queries that require communication with Bing. Table 4 shows the average number of bytes our prototype system sends to and receives from Bing. Since C+K+C queries require the system to send queries twice to the search engine, their traffic is more than C+K queries'. Nonetheless, the communication costs are within reasonable range.

Pattern	sent	received
C + K	270	3177
C + K + C	463	140627

Table 4: Traffic (bytes)

5.2 Offline Precomputation

We precompute a word association matrix using a map-reduce framework and a distributed storage system on a cluster wth 30 ma-

[1] Accessing the document index directly and concurrently, or offering the user the rewritten queries to let he/she click on them, would definitely improve timing.

chines. Each machines has an 8-core 2.33 GHz Intel Xeon E5410 CPU and 16 GB of memory, and is running 64-bit Microsoft Windows Servers 2003. However, our program is single threaded and hence uses only one core at a time on any of the machines.

Pivot words and word association

One assumption we made in this paper is that we can estimate the association between an entity term and a keyword using simple two-way word association. The following experiments verify this assumption.

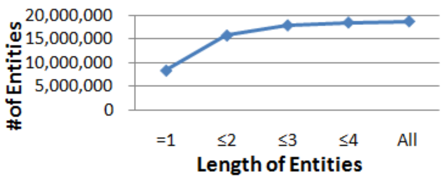


Figure 17: Number of words in Probase entities

We sampled 9,154,141 web pages (25GB on disk) from a web corpus snapshot of 366,185,148 pages for counting co-occurrences of words. We first examine the length of all the entities in Probase. Fig. 17 depicts the distribution of the number of entities in Probase over their lengths (number of words per entity). The result shows that 44.36% of the entities contains just one word, almost 84% have 2 or fewer words, and over 95% have 3 or fewer words. One-word entities, which account for almost half of all entities, are straightforward to process using 2-way word association with another keyword. For the other half of the entities which have more than 1 word, the next experiment indicates that there indeed exists pivot words in many such entities and 2-way association results are very similar to the exact results using multi-way association.

We take the 10 benchmark C+K queries [1], and for each query generate a set of E+K candidate queries by substituting the concept with its associated entities. We next rank this candidate set using two different methods and compare the resulting rankings. In the first method, which serves as a baseline, we rank the E+K queries by the actual association values of all the words in these queries. In other words, we compute the exact multi-way association of the entity term and the keywords by counting the number of times these words co-occur in our web corpus samples. In the second method, we rank the same set of E+K queries by computing 2-way association of the pivot word and the keywords in each query using (4) in Section 4.3.1. To find out the effectiveness of pivot words and 2-way association, we compute the similarity between the two rankings for each of the 10 benchmark queries using *Kendall's Tau* [15]. Kendall's tau distance between two equi-length sequences τ_1 and τ_2 is:

$$\begin{aligned} K(\tau_1,\tau_2) \quad = \quad & |(i,j) : i < j, (\tau_1(i) < \tau_1(j) \wedge \tau_2(i) > \tau_2(j)) \\ & \vee (\tau_1(i) > \tau_1(j) \wedge \tau_2(i) < \tau_2(j))| \end{aligned}$$

Kendall's tau measures the number of "discordant" position pairs, and that number is equal to the number of flip operations required to turn the second sequence into the first sequence by Bubble sort. We further calculate the *normalized Kendall's tau* as:

$$\bar{K}(\tau_1,\tau_2) = 1 - \frac{K(\tau_1,\tau_2)}{n(n-1)/2}$$

Fig. 18 shows that the two rankings are similar enough across all the benchmark queries to warrant the simulation of exact method with the 2-way association with the pivot words. Please see [1] for the the pivot words we discovered in these queries.

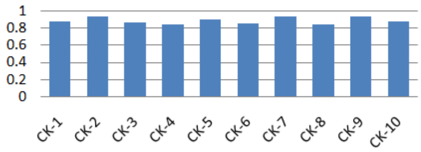


Figure 18: Normalized similarity between ideal and simulated rankings

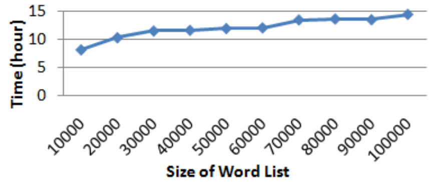


Figure 19: Word association scaling

Time Performance

This experiment evaluates the scalability of word association computation. We varied the length of the word list, or the matrix, from 10,000 to 100,000 and measure the time it takes to compute the $n \times n$ word association matrix on the 30-machine cluster. Fig. 19 shows the roughly linear scale-up graph and indicated that 14.45 hours is required to compute a 100,000 by 100,000 matrix which was the matrix used in the actual prototype system.

6. RELATED WORK

There have been some attempts to support semantic web search using some form of a knowledge base. A well-known example is PowerSet [5] which maintains huge indexes of entities and their concepts. This approach, however, will not scale because updates on the entities can be extremely expensive. Another noteworthy general semantic search engine is Hakia [4] which leverages a commercial ontology and the QDex technology. The QDex technology is unique in that it indexes paragraphs by the embedded frequent queries (sequence of words). Hakia's search results on many of our benchmark queries were similar to keyword search results, which suggest the coverage of their ontology is too limited to help understanding the queries. Other semantic engines include WolframAlpha [7], Evri [3], SenseBot [6] and DeepDyve [2], etc. These engines exploit human curated or automatically extracted knowledge in specific domains such as science, news and research documents. Qiu and Cho proposed a personalized topic search [29] using topic-sensitive PageRank [19], which emphasizes on the analysis of user interests and the disambiguation of entities.

Understanding users' queries and helping users formulate "better" queries is important to document retrieval and web search. Work in this space can be collectively referred to as *query rewriting*. The relevant techniques include query parsing [18], query expansion [26, 33, 17], query reduction [22, 24], spelling correction [13] and query substitution [30, 10]. We next discuss query parsing and query substitution, two techniques that are more closely related to the work in this paper.

One notable example of query parsing which is required for all the other rewriting approaches is Guo *et al.*'s work on named entity recognition in queries, in which they detected entities in queries and identified the most likely classes they belong to. They achieve this through the use of a small human-curated taxonomy and some off-line training, which may not be scalable to various web queries.

Radlinski *et al.* [30] projected web queries to a relatively small set of ad queries, and use the search results to help compute the similarity between two queries. Malekian *et al.* optimized query rewrites for keyword-based advertising [28]. They established a formal graph model to solve the problem. Antonellis *et al.* proposed query rewriting through link analysis of the click graph [10] using SimRank [21] to identify similar queries. However, all of these approaches used a finite ad query set which does not come close to the scale this paper is experimenting with.

Finally, some researchers experimented with indirect assistant to users instead of rewriting the queries automatically. One body of work is providing relevance feedback corresponding to the user's query [35, 9, 8]. Radlinski and Joachims analyzed query chains to help retrieve text results [31]. These methods involve human supervision and requires a second search. Terra and Clarke substitutes query terms from retrieved documents [34]. An initial retrieval for all queries is necessary. In our framework, all query patterns except C+K+C require just one search. Broder and his colleagues [12] uses a taxonomy to classify web queries, but they came short of providing any useful applications for this technique. Jones *et al.* proposed a way to generate query substitutions with the help of search logs [23]. They suggested that the difference between two consecutive queries issued by a same user can be treated as a candidate substitution for one another. Zhang *et al.* improved the above work by using active learning and comparing click logs and editorial labels [36, 37].

7. CONCLUSION

In this paper, we notice a special class of web search which we name it "topic search" and propose an idea to support it. We use an automatically constructed taxonomy to analyze the search log and find that most queries are about entities or concepts. Then we notice that we can use Probase to help us understand the queries and help users to clear their vague concepts. By specifying the concepts into entities and taking advantage of word association information, we do provide additional info for traditional search engines and help them improve the quality of topic search results.

8. REFERENCES

[1] http://research.microsoft.com/en-us/projects/probase/topicsearch.aspx.
[2] Deepdyve. http://www.deepdyve.com.
[3] Evri. http://www.evri.com.
[4] Hakia. http://www.hakia.com.
[5] Powerset. http://www.powerset.com.
[6] Sensebot. http://www.sensebot.net.
[7] Wolfram alpha. http://www.wolframalpha.com.
[8] P. G. Anick. Using terminological feedback for web search refinement: a log-based study. In *SIGIR*, pages 88–95, 2003.
[9] P. G. Anick and S. Tipirneni. The paraphrase search assistant: Terminological feedback for iterative information seeking. In *SIGIR*, pages 153–159, 1999.
[10] I. Antonellis, H. Garcia-Molina, and C.-C. Chang. Simrank++: Query rewriting through link analysis of the click graph. In *VLDB*, June 2008.
[11] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, 2008.
[12] A. Z. Broder, M. Fontoura, E. Gabrilovich, A. Joshi, V. Josifovski, and T. Zhang. Robust classification of rare queries using web knowledge. In *SIGIR*, 2007.
[13] I. Durham, D. A. Lamb, and J. B. Saxe. Spelling correction in user interfaces. *Commun. ACM*, 26(10):764–773, 1983.
[14] H. W. et al. Probase: Building a probabilistic ontology from the web. 2010.
[15] R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. *SIAM Journal on Discrete Mathematics*, 17:134–160, 2003.
[16] C. Fellbaum, editor. *WordNet: an electronic lexical database*. MIT Press, 1998.
[17] B. M. Fonseca, P. B. Golgher, B. Pôssas, B. A. Ribeiro-Neto, and N. Ziviani. Concept-based interactive query expansion. In *CIKM*, pages 696–703, 2005.
[18] J. Guo, G. Xu, X. Cheng, and H. Li. Named entity recognition in query. In *SIGIR*, pages 267–274, 2009.
[19] T. H. Haveliwala. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *IEEE Trans. Knowl. Data Eng.*, 15(4):784–796, 2003.
[20] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *COLING*, pages 539–545, 1992.
[21] G. Jeh and J. Widom. Simrank: a measure of structural-context similarity. In *SIGKDD*, 2002.
[22] R. Jones and D. C. Fain. Query word deletion prediction. In *SIGIR*, pages 435–436, 2003.
[23] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *WWW*, pages 387–396, 2006.
[24] G. Kumaran and V. R. Carvalho. Reducing long queries using query quality predictors. In *SIGIR*, pages 564–571, 2009.
[25] D. B. Lenat and R. V. Guha. *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*. Addison-Wesley, 1989.
[26] M. E. Lesk. Word-word associations in document retrieval systems. *American Documentation*, 20:27–38, 1969.
[27] P. Li and K. Church. A sketch algorithm for estimating two-way and multi-way associations. *Computational Linguistics*, 33(3):305–354, 2007.
[28] A. Malekian, C.-C. Chang, R. Kumar, and G. Wang. Optimizing query rewrites for keyword-based advertising. In *ACM Conference on Electronic Commerce*, 2008.
[29] F. Qiu and J. Cho. Automatic identification of user interest for personalized search. In *WWW*, pages 727–736, 2006.
[30] F. Radlinski, A. Z. Broder, P. Ciccolo, E. Gabrilovich, V. Josifovski, and L. Riedel. Optimizing relevance and revenue in ad search: a query substitution approach. In *SIGIR*, pages 403–410, 2008.
[31] F. Radlinski and T. Joachims. Query chains: Learning to rank from implicit feedback. *CoRR*, 2006.
[32] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, 1988.
[33] G. Salton and C. Buckley. Improving retrieval performance by relevance feedback. *JASIS*, 41(4):288–297, 1990.
[34] E. Terra and C. L. A. Clarke. Scoring missing terms in information retrieval tasks. In *CIKM*, pages 50–58, 2004.
[35] J. Xu and W. B. Croft. Query expansion using local and global document analysis. In *SIGIR*, pages 4–11, 1996.
[36] W. V. Zhang, X. He, B. Rey, and R. Jones. Query rewriting using active learning for sponsored search. In *SIGIR*, 2007.
[37] W. V. Zhang and R. Jones. Comparing click logs and editorial labels for training query rewriting. In *Workshop on Query Log Analysis: Social And Technological Challenges*, 2007.