Searching with Context

Reiner Kraft   Chi Chao Chang

Farzin Maghoul   Ravi Kumar

Yahoo!, Inc.
Sunnyvale, CA 94089, USA
{reiner,chichao,fmaghoul,ravikumar}@yahoo-inc.com

ABSTRACT

Contextual search refers to proactively capturing the information need of a user by automatically augmenting the user query with information extracted from the search context; for example, by using terms from the web page the user is currently browsing or a file the user is currently editing.

We present three different algorithms to implement contextual search for the Web. The first, *query rewriting (QR)*, augments each query with appropriate terms from the search context and uses an off-the-shelf web search engine to answer this augmented query. The second, *rank-biasing (RB)*, generates a representation of the context and answers queries using a custom-built search engine that exploits this representation. The third, *iterative filtering meta-search (IFM)*, generates multiple subqueries based on the user query and appropriate terms from the search context, uses an off-the-shelf search engine to answer these subqueries, and re-ranks the results of the subqueries using rank aggregation methods.

We extensively evaluate the three methods using 200 contexts and over 24,000 human relevance judgments of search results. We show that while QR works surprisingly well, the relevance and recall can be improved using RB and substantially more using IFM. Thus, QR, RB, and IFM represent a cost-effective design spectrum for contextual search.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Storage and Retrieval—*Information Search and Retrieval*

General Terms

Experimentation

Keywords

Contextual Search, Meta-search, Web Search, Specialized Search Engines, Rank Aggregation

1. INTRODUCTION

Current keyword-based search engines have turned out to be remarkably effective so far, especially given the gargantuan Web and the barely cryptic expression of intent by the user via a short web query. Maintaining this effectiveness, however, will be increasingly challenging for two main reasons: an unabated growth of the Web and an increasing expectation placed by the user on the search engine to anticipate and infer his/her information needs and provide relevant results.

Owing to the inherent ambiguity and incompleteness of queries, it is difficult for search engines to maintain high relevance. Queries might be topic-wise ambiguous, as exemplified by "jaguar" or "Michael Jordan." Queries might be intent-wise ambiguous; for example, "stem cell" might connote either biology or ethics. Interestingly, disambiguation is immediate if one pays attention to the context, for example, that the user queried for "jaguar" while browsing an automotive review site or queried for "stem cell" while reading a US senate briefing. This is precisely the theme behind contextual search. In a nutshell, contextual search refers to proactively capturing the information need of a user by automatically augmenting the user query with information extracted from the search context; for example, by using terms from the web page the user is currently browsing or a file the user is currently editing.

The reason why contextual search can be very effective is quite intuitive. Search queries are often formulated while the user is engaged in some larger task. In these cases, the context is bound to contain information that can help refine the meaning of the user's query. A contextual search engine—such as Y!Q Contextual Search [18]—might take the context as an additional input to disambiguate and augment the user's explicit query. Thus, in addition to disambiguating queries, context can help narrow searches and reveal latent user intent. In fact, a study of query logs shows that many users already do some form of contextual search, albeit manually [20]. They use additional words to refine and reissue queries when the search results for the initial query turn out to be unsatisfactory; these additional words are often obtained from the browsing context. Unfortunately, user-initiated contextual search in the form of query refinement has its obvious limitations, necessitating an automatic means for searching with context.

Implementing contextual search involves two tasks. The first is the user interface, which has been extensively discussed within the context of Y!Q [18]. The second consists of extracting and representing the context, and using the context in conjunction with the query. This second task the main focus of our paper.

We provide a spectrum of algorithms to perform automatic contextual search and present three of them—query

rewriting, rank-biasing, and iterative filtering meta-search—that address a variety of engineering and relevance needs, from the most efficient to the most effective.

The first algorithm, *query rewriting (QR)*, augments each query with appropriate terms from the search context and uses an off-the-shelf web search engine to answer this augmented query. The second algorithm, *rank-biasing (RB)*, generates a representation of the context and answers queries using a custom-built search engine that exploits this representation. The third algorithm, *iterative filtering meta-search (IFM)*, generates multiple subqueries based on the user query and appropriate terms from the search context, uses an off-the-shelf search engine to answer these subqueries, and re-ranks the results of the subqueries using rank aggregation methods.

QR is an attractive method because it fits naturally with the "querybox" semantics offered by major search engines, and as such can be implemented on top of them in a straightforward fashion. The semantic engine basically suggests query terms to be added to the query box. Analogously, most web search users typically add more query terms when reformulating their queries in an attempt to get more relevant results [20]. RB departs from this natural model and takes advantage of a special feature—the ability to boost results using contextual information that may or may not be available in major search engines. IFM relies on meta-search and rank aggregation in an all-out attempt to deliver the most relevant results.

We then systematically study the three algorithms by using 24,566 judgments from 28 expert judges; these judgments arose out of a benchmark of 200 contexts. Our findings are summarized below:

- QR performs surprisingly well and has different optimal operating points depending on the web search engine used. In particular, the operating points are a function of the effective recall.

- RB and IFM mitigate the recall limitations encountered with QR. IFM is very effective and outperforms the others both in recall and relevance.

- Human reformulations are very unlikely to achieve comparable relevance with respect to any of the three methods.

In Section 2, we review related work and provide more background on contextual search, clarifying the terminology used in the remainder of the paper. In Section 3, we present the three different algorithms for contextual search. In Section 4, we describe the experimental setup and metrics, and present the evaluation results.

2. PRELIMINARIES

This section discusses related work and defines the terminology we will use throughout the remainder of this paper.

2.1 Related Work

The majority of the contextual search work revolves around learning user profiles based on previous searches, search results, and (recently) Web navigation patterns. The information system uses this learning to represent the user interest for the refinement of future searches. Another area of learning is focused on context learning (e.g., [15, 2]) based on judged relevant documents, query terms, document vectors, etc.

"Context as a query" (e.g., [16, 23, 6, 8, 4]) treats the context as a background for topic specific search and extracts the query representing the context and therefore, the task at hand. Some recent contextual search tools (e.g., Y!Q [18] (`http://yq.search.yahoo.com`), Blinkx (`http://www.blinkx.com/`), The Dashboard (`http://nat.org/dashboard/`), IntelliZap [14]), and more general contextual ads (e.g., Google AdSense (`https://www.google.com/adsense/`), Yahoo! Publisher Network (`http://publisher.yahoo.com`)) explore this idea further. Lawrence [21] argues that next generation search engines will increasingly make use of context information.

The IFM approach can be used as a basis to build *Web carnivores*. Etzioni coined this colorful phrase [10]. In this analogy, web pages are at the bottom of the Web information food chain. Search engines are the *herbivores* of the food chain, grazing on web pages and regurgitating them as searchable indices. Carnivores sit at the top of the food chain, intelligently hunting and feasting on the herbivores. The carnivore approach leverages the significant and continuous effort required to maintain a world-class search engine (crawling, scrubbing, de-spamming, parsing, indexing, and ranking.) In a search context, the carnivore approach is applicable when standard web search engines are known to contain the documents of interest but do not return them in response to simple queries.

2.2 Terminology

We stated earlier that the relevance of web search results can be improved if context is used. Unfortunately the word "context" is overloaded with many different meanings. We therefore want to state more precisely what we mean by *context* and give some related terminology.

Context, in its general form, refers to any additional information associated with the query. In this paper we narrow context to represent a piece of text (e.g., a few words, a sentence, a paragraph, an article) that has been authored by someone.

This naturally leads to a *context term vector*. This is a dense representation of a context that can be obtained using various text or entity recognition algorithms (e.g., [5, 3, 25]) and represented in the vector space model [26]. In this model extracted terms are typically associated with weights that represent the importance of a term within the context, and/or additional meta-data. The terms of a context term vector may represent (but are not limited to) a subset of the words/phrases/entities in the content of the context. For completeness we point out that there are contextual search implementations (e.g., ConQuery[1]) that do not make use of a context term vector, but rather use the selected context itself as part of the query. There are relevancy problems associated with such an implementation, since the selected text may contain many noise terms, stop-words, and terms that are not central to the gist of the context.

A search query that comprises a keyword query and a context (represented by a context term vector) is called a "contextual search query". Notice that either of these components can be empty. If the keyword query is empty, then it is called "query-less." *Contextual search* refers to a search metaphor that is based on contextual search queries.

---

[1]http://conquery.mozdev.org/

We identify two families of queries. *Simple queries (SQ)* are regular keyword based search queries, i.e., not contextual search queries. They typically comprise a few keywords or phrases, but no special or expensive operators (e.g., proximity operators.) *Complex queries (CQ)* are regular keyword based search queries and typically consist of keywords or phrases plus ranking operators; they are more expensive to evaluate.

Standard search engine (Std. SE) refers to web search engines like Yahoo! (`http://search.yahoo.com`) or Google (`http://www.google.com`) that support simple queries. A *modified search engine (Mod. SE)* refers to a web search engine that has been *modified* to support complex search queries using rank-biasing operators. Finally, a *contextual search engine (CSE)* is an application front-end that supports contextual search queries.

In the remainder of the paper we use the term "query" to refer to regular keyword search queries. If we refer to a "contextual search query", we will explicitly state so.

3. CONTEXTUAL SEARCH

As we pointed out earlier, there are different methods to implement contextual search. Figure 1 illustrates these. There are two dimensions that can characterize a particular method. The first is the number of queries we send per contextual search query to the search engine, either sequentially or in parallel. The second is the *type of queries* we send to the search engine; we distinguish here between *simple* and *complex* queries (see earlier definition.)

We can enumerate and name these methods as follows (the numbering corresponds to Figure 1):

(1) **Query Rewriting (QR):** Send one simple query per contextual search query to a standard search engine.

(2) **Rank-Biasing (RB):** Send one complex query per contextual search query to a modified search engine.

(3) **Iterative, Filtering Meta-Search (IFM):** Send multiple, simple queries per contextual search query to a standard search engine.

For completeness we also mention the fourth method of sending multiple, complex queries to a modified search engine backend. However, we did not consider this case further for the following reason. Within a search engine architecture, given the limited resources, there are economic decisions to make when trying to increase its scalability to support more traffic. One way to increase scalability is to improve the number of simple queries that can be handled per second; this is the typical scenario. The other way is to allow queries to be more complex, i.e., to put in more logic into the ranking function. However, investing in *both* scenarios is quite expensive from a practical standpoint.

3.1 Query Rewriting

In this scenario (Figure 1), for every contextual search query, we send one, simple query to a standard search engine backend. The query rewriting approach creates queries composed of all query and context term vector terms to form a—possibly rather long—query using `AND` semantics.

QR takes as a parameter the number of terms taken from the context term vector that should be concatenated with the query. In the experimental section (Section 4) we describe QR1 (that takes only the top ranked term from the context term vector), QR2 (that takes the top two terms from the context term vector), and so on up to QR5 (that takes the top five terms from the context term vector.)

An example illustrates how QR generates queries. Consider the query `q` and a context term vector comprising the terms `(a,b,c,d,e,f)` ranked in decreasing order of their weight. Based on this input QR1 will construct the query `q a`, QR2 will construct `q a b`, and QR5 construct will construct `q a b c d e`.

The major advantage of this approach is simplicity, especially since conjunctive semantics is supported in all major search engines. The disadvantages are:

- The more terms are added in a conjunctive way, the more restricted the query is, and the less results it will likely return, i.e., there is a danger of low recall.

- The query and context term vector together may comprise more terms than the search engine supports for evaluation.

3.2 Rank-Biasing

This approach requires a modified search engine back-end with rank-biasing operators and capabilities. This allows the query generator component to generate and send a complex query that contains ranking instructions to such a modified search engine backend (see Figure 1). The main goal is to have the same level of recall as if the query were issued without a context.

Each query is made up of two parts, the selection part effecting recall, and optional ranking terms only impacting the score of the selected documents. For example,

`<query> = <selection=cat> <optional=persian, 2.0>`

This selects all documents containing the term "cat". However, it boosts the score of those selected documents containing the terms "persian" by a factor of 2.0. The weight associated to each optional term determines how strong the term should influence the overall ranking.

One approach for implementing contextual search using the rank-biasing approach is to use the selection part for representing the regular query, and the optional ranking terms to represent extracted terms from the context.

RB takes as input the following parameters:

Number of selection terms to use: This parameter determines the number of top ranked extracted terms from the context term vector that are being used as selection terms using conjunctive semantics.

Number of rank operators: RB requires support for a special `RANK` operator that boosts the ranking of a document that contains the desired term. This parameter determines how many `RANK` operators should be used. Those `RANK` operators are appended to the rewritten query (after the selection terms that have been appended.)

Weight multiplier for each `RANK` operator: Recall that a term within a context term vector has associated weights. This parameter is a scaling parameter that adjusts the weights of the terms in the context term vector to match the scaling of the search engine's ranking internals.
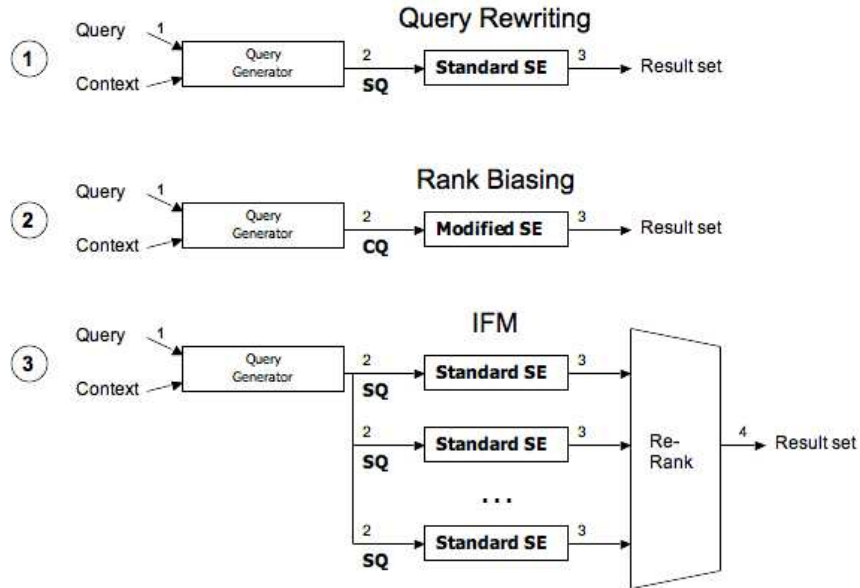
Figure 1: Overview of the three methods for contextual search: query rewriting, rank-biasing, and IFM.

An example illustrates how RB generates queries. Consider a query `q` and a context term vector comprising the terms `(a:100,b:90,c:80,d:70,e:60,f:50)` ranked in decreasing order of their weight (now we have term-weight pairs). Based on this input, an RB using 2 context terms as selection terms, 2 `RANK` operators, and a weight multiplier of 0.1 constructs a query as follows:

`q a b RANK(c,8.0) RANK(d,7.0)`

Since we are using two rank operators in this example, we refer to this configuration as RB2.

Our underlying assumption and motivation for this approach is to boost the ranking score of the matching documents that are related to the context. The main disadvantage of this method is the need for a specialized search engine for back-end support.

3.3 Iterative Filtering Meta-search

Overall the IFM model (see Figure 1) is based on the concept of meta-search. Meta-search engines were among the first applications built on top of web search engines. They distribute their queries to possibly multiple "real" search engines and combine the results to the user. SavvySearch [17] and MetaCrawler [24] were both launched in mid-1995 and appear to have been invented independently. Since then, a large literature has been written on the topic (see [22] for a survey.)

There are two major components in the implementation of IFM: query generation and ranking/filtering. The first component deals with the generation of subqueries corresponding to every contextual search and the second component deals with aggregating the results of these subqueries. In the following sections we describe these two components in further detail.

Recall that the main idea in IFM is to explore the problem domain in a systematic way by sending many appropriate subqueries to the search engine. Each subquery can be seen as a different strategy that should be explored. The task is then to aggregate the results of the subqueries; this aggrega-

tion step will involve filtering and re-ranking of the results of the subqueries.

3.3.1 Query Generation

For generating subqueries, we use the *query template* approach; this is similar to the one adopted in the buying guide finder [19]. Templates specify what subqueries to construct from the pool of terms. Given a query and a context, the first step is use these to derive a candidate pool of terms and a context term vector. The template is then applied to this pool to generate many possible subqueries.

As an example, suppose we have a query `q` and a context term vector `(a,b,c,d)`, where the terms are ranked by decreasing weight. A query template might suggest generating the following subqueries: `q a, q b, q c, q d, q a b, q b c ,q c d, q a b c, q a b d`, and `q a b c d`.

Query templates thus represent a convenient means to specify what type of subqueries to construct from the given pool of terms, since it is often prohibitive to exhaustively enumerate all possible subquery combinations; it suffices to pick only those subqueries that are most viable to produce good results.

For the purposes of this paper, we experimented with different query templates. A particularly interesting template is a sliding window of different sizes over the terms in the context term vector. In the above example, if the window size is 2, the template would then be `q a b, q b c`, and `q c d`. We exposed the size of the sliding window as a parameter. In the experimental section we introduce IFM-SW1 (uses a sliding window of size 1), IFM-SW2 (uses a sliding window of size 2), up to IFM-SW4 (uses a sliding window of size 4.)

We also considered a different template where the subquery is forced to contain the higher ranked terms in the context along with the actual query, while the remainder of the subquery is made up of the lower ranked terms from the context, in different permutations. Again, in the above example, if `a` and `b` are deemed to be of high rank, a real-

ization of this template might be of the form `q a b c`, `q a b d`, and `q a b c d`.

As we will see later, our experiments indicate that a very small and carefully chosen template is sufficient to produce good results in practice.

3.3.2   Ranking and Filtering

At the end of the query generation phase, we issue the k subqueries to the standard SE and obtain the results for each of the k subqueries. The challenge now is how to combine, re-rank, and filter the results of the subqueries in a meaningful way.

At this point, we note that this phase was relatively easier in the buying guide finder application [19]. There, a doc-type classifier was available to label whether a document is a buying guide or not—a simple binary value. Our present setting is clearly more complicated since it is desirable that the combining mechanism exploits the actual ranking of the results. To accomplish this, we resort to the paradigm of rank aggregation [24, 9], which is often used in meta-search and meta-crawlers.

Rank aggregation is the problem of combining several ranked lists in the best way possible into a single ranked list. While there are many approaches to rank aggregation, a natural way is to cast it in a simple combinatorial optimization framework; for the remainder of the discussion, we will take this route. In the most idealized form, we are given a universe U and k ranked lists (i.e., permutations) $\pi_1, \ldots, \pi_k$ on the elements of the universe. The goal is to combine these k lists into a single ranked list π^* such that $\sum_{i=1}^{k} d(\pi^*, \pi_i)$ is minimized. Here, the distance function $d(\cdot, \cdot)$ can be any metric on permutations. Popular choices include the Spearman footrule metric $F(\pi_1, \pi_2) = \sum_{u \in U} |\pi_1(u) - \pi_2(u)|$, which resembles the L_1 distance and the Kendall tau metric $K(\pi_1, \pi_2) = \sum_{u,v \in U} [\pi_1(u) > \pi_1(v) \wedge \pi_2(u) < \pi_2(v)]$, which counts pairwise disagreements. From a computational point of view, the choice of the actual metric is critical. If d is set to be the footrule metric, then the optimization problem can be solved exactly in polynomial time; this can be done by minimum cost perfect matching algorithms. If d is set to be the Kendall metric, then the optimization problem becomes NP-hard. However, very good approximation algorithms exist for the problem. For a description of the algorithms and some applications, the readers are referred to [9] and [13, 12].

Rank aggregation is a natural candidate to use in our setting because it is a principled approach to combine several ranked lists where there is a clear notion of a defensible objective function and the aggregation methods work to optimize the objective. There are two caveats using rank aggregation, however. The first is that many rank aggregation methods operate in a "garbage-in-garbage-out" mode, i.e., their results might not be impressive if the input lists are highly non-overlapping. However, this is not a concern for us since our query generation phase has a built-in expectation that there would be overlap among the results for subqueries. The second caveat is the assumption that the input lists are permutations; clearly this assumption does not hold true in our setting since the top 100 results of the subqueries need to be permutations of one another. Fortunately, many of the rank aggregation algorithms can be modified to handle this "partial list" case.

We experimented with two basic algorithms for rank aggregation [9]. Recall that the input is k ranked lists, which are the top few results of k subqueries. The first algorithm, called *rank averaging*, works by assigning a score to every position in a rank list, say, 1 for the first position, 2 for the second position, and so on. Then, the score of a URL is the average of the scores it obtains in each of the input lists. If a URL is not present in a list, its score is size of the list plus 1. The final ordering of URLs is obtained by sorting them according to their average scores; ties are broken arbitrarily. This method is computationally very attractive and is an adaptation of the popular voting method called Borda method. Borda method is a constant-factor approximation to finding the optimal ranking with respect to the Kendall metric [11].

The second algorithm, called *MC4*, is more sophisticated, computationally intensive, and is based on Markov chains. The idea is to construct a Markov chain, where the states of the chain are the URLs in the union of the k lists and a transition is added from URL u to URL v if a weighted majority of the input lists rank u above v. Notice that if a URL is not present in the top results of a list, it is considered to be ranked lower than all the URLs present in the list. After making the Markov chain ergodic and aperiodic by adding a small transition probability from every URL to every other URL, we compute the stationary distribution of the chain. The stationary probabilities then determine the ranking of the URLs. It was shown experimentally that MC4 performs extremely well in practice [9, 13].

4.   EVALUATION AND RESULTS

4.1   Experimental Setup

We use a test benchmark of 200 contexts. The Y!Q Contextual Search interface [18] is instrumented to log the contexts selected or highlighted by users. These contexts are mined out of the Y!Q web logs and sampled to arrive at the final benchmark we used as a basis for our experiments. The minimum length of a context in that benchmark was 19 words, the average was 338 words, and the maximum length was 1991 words. The average context term vector size was 13 entries, the smallest was 1 entry and the largest was 15 entries. Six context term vectors out of 200 had fewer than 5 vector entries. Recall in Figure 1 contextual search could take as input arguments a query and a context. In our experiments we only use context (query is nil.)

We tested a total of 41 configurations:

- 15 configurations of QR. We used three different web search engines—Yahoo, MSN, and Google—for each of the five QR configurations as described in Section 3.1.

- 18 configurations of RB. We used 1 and 2 context terms as part of the query; 2, 4, and 6 `RANK` operators; and three different weight multipliers: 0.01, 0.1 and 0.5. We use Yahoo as the web search engine. However, we only report the two best RB configurations.

- 8 configurations of IFM. We experimented with rank averaging and MC4 algorithms, using Yahoo as the web search engine as well. For each algorithm, we tried sliding window sizes of 1, 2, 3, and 4.

Table 1 presents a summary of the 25 configurations reported in this paper. Notice that as the number of context

Configuration	Description	Web Search Engines
QR1	average of 2.25 terms	Yahoo, MSN, Google
QR2	average of 4.14 terms	Yahoo, MSN, Google
QR3	average of 5.73 terms	Yahoo, MSN, Google
QR4	average of 7.30 terms	Yahoo, MSN, Google
QR5	average of 8.61 terms	Yahoo, MSN, Google
RB2	2 rank operators, weight multiplier= 0.1 1 context terms in query	Yahoo
RB6	6 rank operators, weight multiplier= 0.01 2 context terms in query	Yahoo
IFM RA	IFM using rank averaging	Yahoo, 4 sizes of sliding window (SW)
IFM MC4	IFM using MC4	Yahoo, 4 sizes of sliding window (SW)

Table 1: Summary of the 25 configurations reported.

vector entries increases, the average number of query terms in QR increases from 2.25 terms (QR1) to 8.61 terms (QR5). Meanwhile, the average query length of reformulated queries (obtained from Yahoo logs in January of 2005) is 2.922 words (95 % confidence interval is [2.873, 2.972]), compared to an average of 2.54 ([2.45, 2.66]) based on ComScore (which includes MSN, Google, and Yahoo) data (http://www.comscore.com). This suggests that an idealized QR configuration that could emulate human search behavior would be somewhere between QR1 and QR2.

4.2 Relevance Methodology

In a standard Cranfield model [7], judgments are issued with respect to a query in a blind setting to remove bias. Our evaluation is "end-to-end", that is, we are only concerned with whether a contextual search configuration exhibits a boost in overall relevance and not with the quality of intermediate data such as context vectors. Our methodology differs from the standard framework in two ways:

Relevancy to the Context: Judges are asked to provide relevance judgments for documents with respect to the context. They are asked to read the context carefully prior to issuing any judgment.

Perceived Relevance: Search engines always return document titles and abstracts that are highly tailored to the given query (recall that query here refers to SQ and CQ queries constructed from the original query and the context). In our setting, judges base their relevance judgment on the title and abstract of the document, instead of the content of the landing page. Automatic document summarization engines have improved substantially with time; generally speaking, any of the three major web search engines used in testing does a good job distilling the gist of the document. Perceived relevance judgments are cost-effective and keep the same level of sensitivity and resolution compared to judgments based on contents of the entire document [1].

A judge is asked to select a context that he or she feels most comfortable or least uncomfortable out of the pool of 200 contexts. Once selected, the context is removed from the pool. Given a context, a judge is presented with a list of web results, one at a time. For each web result, the judge answers the following question, *"Is this result relevant to the context?"* Judgments are captured in a 3-point scale: *"Yes"*, *"Somewhat"*, *"No"*, and *"Can't Tell"*. A result is relevant when it bears a strong connection to the context beyond the superficial; it often times offers additional, complementary, and related information to the context, generally leading to a satisfactory search experience. A result is somewhat relevant when it merely repeats the information provided by the context, or when it is relevant to only one aspect or meaning of the context. An irrelevant result is one that has no bearing to the context even if it merely matches on some keywords. Finally, a judge can decide to bypass judgment if he/she simply cannot tell whether the result is relevant or not.

Consider the following context:

> Cowboys Cut Carter; Testaverde to Start OX-NARD, Calif Quincy Carter was cut by the Dallas Cowboys on Wednesday, leaving 40-year-old Vinny Testaverde as the starting quarterback. The team wouldn't say why it released Carter.

This context discusses at length the Dallas Cowboys' (an American pro-football team) decision to release quarterback Quincy Carter. The closing paragraph mentions the fact that this action leaves Vinny Testaverde as the starting quarterback. A web result relating directly to the Dallas Cowboys (i.e., the official website with news flashes) and Quincy Carter would typically be judged *"Yes"*. A result that is apparently repeating the same as or very similar information to the context may be judged *"Somewhat"*. A result about Jimmy Carter, the former president of the U.S., would be judged *"No"*. A result entitled "Cowboy History and Information" with an abstract that does not clarify whether this refers to Dallas Cowboys or an ordinary cowboy would typically receive a *"Can't Tell"*. To evaluate relevance, we collected relevance judgments from 28 expert judges for the top 3 results returned by each of these 41 configurations. A total of 24,556 judgments were entered over a period of three weeks.

4.3 Coverage Results

Table 2 shows the average and median number of results returned using the QR configurations for each web search engine. There is a substantial drop in recall as number of vector entries in QR increases: the drop is comparable between MSN and Yahoo whereas it is roughly one order of magnitude less pronounced with Google.

Configuration	MSN	Yahoo	Google	MSN	Yahoo	Google
QR1	1,511,351.50	405,833.50	7,003,387.50	63,443.00	21,954.50	362,000.00
QR2	121,585.69	40,888.04	1,053,351.38	791.00	575.00	1,070.00
QR3	3,216.58	2,531.92	24,636.72	116.00	59.50	325.00
QR4	872.67	578.10	8,437.63	18.00	25.00	245.00
QR5	473.33	363.97	3,349.08		12.00	81.00

Table 2: Average (columns 1 to 3) and median number (columns 4 to 6) of results returned using the QR configurations for each backend engine.

Configuration	MSN zero	< 3	Yahoo zero	< 3	Google zero	< 3
QR1	0 %	0 %	0 %	0 %	0 %	0 %
QR2	0 %	1 %	0 %	3 %	0 %	0 %
QR3	1 %	11 %	1 %	11 %	3 %	4 %
QR4	6 %	21 %	6 %	20 %	4 %	7 %
QR5	9 %	28 %	9 %	26 %	5 %	12 %

Table 3: Coverage drop, define as the percentage of contexts for which there are no web results (zero) and fewer than 3 web results for various QR configurations using MSN, Yahoo, and Google.

To introduce another perspective, Table 3 shows the percentage of contexts for which there are no web results and fewer than 3 web results for QR using MSN, Yahoo, and Google. For QR4 using MSN and Yahoo, low recall could potentially affect user experience as over 20% of context would return fewer than 3 web results. The recall of the RB configurations tested is the same as that of QR2.

What about IFM? In theory, it would be size of the union of all results fetched by the sliding window queries. Table 4 shows the estimated lower bound for this quantity— computed as the maximum number of results returned by any one of these queries—averaged over all 200 contexts. It shows the estimated for SW1 to SW4 configurations, which use as a query template a sliding window of the respective size. Clearly, IFM operates on a much larger set of candidate results that its corresponding QR.

4.4 Relevance Results

For our experiments we are using the following metrics:

Precision at 1 and 3 (P@1, P@3): Defined as the number of relevant results divided by the number of retrieved results, but capped at one (or three), and expressed as a ratio. A result is considered relevant if it receives a judgment of *Yes* or *Somewhat* for Precision.

Strong Precision at 1 and 3(SP@1, SP@3): Defined as the number of relevant results divided by the number of retrieved results, but capped at one (or five), and expressed as a ratio. A result is considered relevant if it receives a judgment of *Yes-only* for Precision.

We look at precision metrics at position 1 and cumulatively at position 3 because the Y!Q overlay GUI displays at most three results.

Tables 5 and 6 show SP and P metrics for the QR configurations, respectively, using the three engines. Unlike Google, notice that SP drops sharply for both MSN and Yahoo beyond QR4 due to the low recall. The drop at position 1, around 4 basis points, is statistically significant and would

likely affect user experience. To some extent, the same effects are observed with precision. The operating points for optimal relevance for MSN and Yahoo backends are QR3 or QR4 depending on the metric to be optimized, while Google still performs well at QR5 for both metrics.

Table 7 shows SP and P for RB and IFM configurations. RB2 and RB6 achieved the best relevance out of all 18 RB configurations. RB-2 recorded the highest SP@1; In general, RB (and QR) appear to do a good job fetching highly relevant results at position 1 for a number of contexts. To gain some insights, roughly 1 out of every 3 URLs returned by RB6 (weight 0.01) for a context came from ranks 100 to 1000. RB2 (weight 0.1) was able to retrieve 2 out of three from the deep set. These results are hit or miss, that is, they are more likely to receive a *Yes* judgment than a *Somewhat* one, hence higher SP but lower P. IFM-RA-SW3 recorded the best precision 0.887 at position 1 amongst *all* configurations. IFM brings up more *Somewhat* relevant results and yields better overall precision. There is little difference between RA and MC4; for both, SW3 produces the best precision metrics and issues on average 2 to 4 queries, which beat our early expectations.

4.5 Discussion of Results

QR, using a highly tuned semantic engine, can attain high relevance. We compared the precision of QR implemented on top of three major search engines and saw that relevance can be affected by low recall for long queries; in fact, precision decays as a function of low recall. However, the optimal point depends on the web search engine and QR can be configured competitively for any of the major three engines. For example, QR3 for MSN attains a P@1 of 0.80, QR4 attains a P@1 of 0.813 and 0.828 for Yahoo and Google respectively. It is important to point out low recall can be attributed to various reasons such as query processing logic, ranking policies, and index size.

Human reformulations are unlikely to attain the same level of relevance as that of QR. The best tested QR1, which issues an average of 2.25 terms per query and would be in

IFM Configuration	Yahoo
SW1	6,817,900
SW2	2,900,682
SW3	244,193
SW4	22,229

Table 4: Estimated lower bound for IFM's recall, computed as the maximum number of results returned by any one of these queries, averaged over all 200 contexts.

Configuration	MSN		Yahoo		Google	
	SP@1	SP@3	SP@1	SP@3	SP@1	SP@3
QR1	0.259	0.250	0.255	0.250	0.284	0.254
QR2	0.405	0.364	0.390	0.375	0.411	0.384
QR3	**0.416**	0.390	**0.423**	0.397	0.435	0.395
QR4	0.399	**0.396**	0.412	**0.416**	0.432	0.394
QR5	0.364	0.358	0.365	0.394	**0.441**	**0.404**

Table 5: SP metrics for the QR configurations, respectively, using the three backends.

Configuration	MSN		Yahoo		Google	
	P@1	P@3	P@1	P@3	P@1	P@3
QR1	0.503	0.504	0.531	0.496	0.513	0.489
QR2	0.728	0.687	0.677	0.688	0.726	0.717
QR3	**0.800**	0.770	0.784	0.758	0.793	0.784
QR4	0.792	**0.775**	**0.813**	**0.801**	**0.828**	0.801
QR5	0.775	0.757	0.798	0.780	0.812	**0.802**

Table 6: P metrics for the QR configurations, respectively, using the three backends.

Configuration	SP@1	SP@3	P@1	P@3
RB2	**0.450**	**0.390**	0.803	0.742
RB6	0.413	0.358	0.755	0.684
IFM RA SW1	0.243	0.253	0.524	0.502
IFM RA SW2	0.399	0.364	0.803	0.730
IFM RA SW3	0.395	0.372	**0.887**	**0.794**
IFM RA SW4	0.366	0.357	0.855	0.785
IFM MC4 SW1	0.251	0.255	0.503	0.497
IFM MC4 SW2	0.354	0.354	0.797	0.721
IFM MC4 SW3	0.370	0.367	0.870	0.787
IFM MC4 SW4	0.363	0.337	0.845	0.762

Table 7: SP and P for RB and IFM configurations.

the low end compared to the average length of reformulated queries (2.92 terms), attains a P@3 of 0.504. The best QR2, with 4.14 terms in average and hence an upper bound to human reformulation, attains a P@3 of 0.717, which is still far lower than the optimal configurations above.

Results show that RB can perform competitively, in particular at the top position. It achieves SP@1 of 0.45, which is the highest measured of all configurations. Further insights indicate that some results that contributed to the high precision at position 1 has been bubbled up from the middle-tier of results (100-1000). However, RB does not do well in overall precision as measured by P@3 (between 0.684 and 0.742). One explanation is that if the "right" results are not recalled by the simple query, bubbling others to the top may be detrimental to overall relevance. To get a sense of the effect of each RB parameter, we conducted a factorial analysis on them. The number of rank operations had no significant impact in relevance (in fact, the best ones had 2 and 6 rank operators) whereas number of terms and weights, when treated as combined factors, had some (Generally for low terms we should use high weights.) Given that RB would require substantial modifications to a web search engine, contextual search services will unlikely pursue this route. However, if a web search engine already has support for rank-biasing operators the overall effort decreases and the remaining work boils down to tuning the weights of the rank operators to support contextual search optimally.

So far, our results suggest that contextual search is not solely a ranking problem, but one of recall. We pursued IFM to leverage meta-search capabilities to improve recall substantially. IFM uses ranking aggregation mechanisms such as the presented *rank averaging* and *MC4* to combine and rank the results. Our experiments demonstrate that:

- IFM indeed achieves an effective recall that is an order of magnitude higher than that of QR and RB;

- IFM can be competitive and, in some measures, superior to QR. IFM can be configured to achieve a P@1 of 0.887, which is substantially higher than 0.813 and 0.803 achieved by QR4 and RB2 (using the same backend engine) respectively. The highest P@3 for IFM is clocked at 0.794, which is comparable to the 0.801 achieved by QR4.

5. CONCLUSIONS

We investigated three methods—QR, RB, and IFM—for contextual search. We showed that QR, a simple technique that closely emulates human query reformulation and can be easily implemented on top of a commodity search engine, performs surprisingly well and is likely to be superior to manual reformulation. RB and IFM break the recall limitations of QR: IFM is very effective and outperforms the others both in terms of recall and relevance at an added engineering cost. These three techniques offer a good design spectrum for contextual search implementors.

Acknowledgments

6. REFERENCES

[1] K. Ali, C. Chang, and Y. F. Juan. Exploring cost-effective approaches to human evaluation of search engine relevance. In *Proceedings of the 27th European Conference on Information Retrieval (ECIR)*, pages 360–374, 2005.

[2] N. J. Belkin, R. Oddy, and H. M. Brooks. ASK for Information Retrieval: Part I. Background and Theory, Part II. Results of a Design Study. *Journal of Documentation*, 38(3):61–71, 145–164, 1982.

[3] D. M. Bikel, R. L. Schwartz, and R. M. Weischedel. An algorithm that learns what's in a name. *Machine Learning*, 34(1-3):211–231, 1999.

[4] D. Billsus, D. Hilbert, and D. Maynes-Aminzade. Improving proactive information systems. In *Proceedings of the 10th International Conference on Intelligent User Interfaces (IUI)*, pages 159–166, 2005.

[5] A. Borthwick, J. Sterling, E. Agichtein, and R. Grishman. Exploiting diverse knowledge sources via maximum entropy in named entity recognition. In *Proceedings of the 6th Workshop on Very Large Corpora*, 1998.

[6] J. Budzik and K. Hammond. Watson: Anticipating and contextualizing information needs. In *Proceedings of the 62nd Annual Meeting of the American Society for Information Science (ASIS)*, pages 727–740, 1999.

[7] M. K. C.W. Cleverdon, J. Mills. Factors determining the performance of indexing systems. *Volume I - Design, Volume II - Test Results, ASLIB Cranfield Project, Reprinted in Sparck Jones & Willett, Readings in Information Retrieval*, 1966.

[8] M. Czerwinski, S. Dumais, G. Robertson, S. Dziadosz, S. Tiernan, and M. van Dantzich. Visualizing implicit queries for information management and retrieval. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pages 560–567, 1999.

[9] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proceedings of the 10th International Conference on World Wide Web (WWW)*, pages 613–622, 2001.

[10] O. Etzioni. Moving up the information food chain: Deploying softbots on the world wide web. In *Proceedings of the 13th National Conference on Artificial Intelligence and the 8th Conference on Innovative Applications of Artificial Intelligence*, pages 1322–1326, 1996.

[11] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee. Rank aggregation: An algorithmic perspective. Manuscript, 2005.

[12] R. Fagin, R. Kumar, K. S. McCurley, J. Novak, D. Sivakumar, J. A. Tomlin, and D. P. Williamson. Searching the workplace web. In *Proceedings of the 12th International Conference on World Wide Web (WWW)*, pages 366–375, 2003.

[13] R. Fagin, R. Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 301–312, 2003.

[14] L. Finkelstein, E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin. Placing search in context: the concept revisited. *ACM Transactions on Information Systems*, 20(1):116–131, 2002.

[15] A. Goker. Capturing information need by learning user context. In *Proceedings of the 16th International Joint Conference in Artificial Intelligence: Learning About Users Workshop*, pages 21–27, 1999.

[16] M. Henzinger, B.-W. Chang, B. Milch, and S. Brin. Query-free news search. In *Proceedings of the 12th International World Wide Web Conference (WWW)*, pages 1–10, 2003.

[17] A. E. Howe and D. Dreilinger. SAVVYSEARCH: A metasearch engine that learns which search engines to query. *AI Magazine*, 18(2):19–25, 1997.

[18] R. Kraft, C. C. Chang, and F. Maghoul. Y!Q: Contextual search at the point of inspiration. In *Proceedings of the 14th Conference on Information and Knowledge Management (CIKM)*, pages 816–823, 2005.

[19] R. Kraft and R. Stata. Finding buying guides with a web carnivore. In *Proceedings of the 1st Latin American Web Congress (LA-WEB)*, pages 84–92, 2003.

[20] R. Kraft and J. Zien. Mining anchor text for query refinement. In *Proceedings of 13th International Conference on World Wide Web (WWW)*, pages 666–674, 2004.

[21] S. Lawrence. Context in web search. *IEEE Data Engineering Bulletin*, 23(3):25–32, 2000.

[22] W. Meng, C. Yu, and K.-L. Liu. Building efficient and effective metasearch engines. *ACM Computing Surveys*, 34(1):48–89, 2002.

[23] B. Rhodes and T. Starner. The remembrance agent: A continuously running automated information retrieval system. In *Proceedings of 1st International Conference on the Practical Application of Intelligent Agents and Multi Agent Technology (PAAM)*, pages 487–495, 1996.

[24] E. Selberg and O. Etzioni. The MetaCrawler Architecture for Resource Aggregation on the Web. *IEEE Expert*, 12(1):8–14, 1997.

[25] R. Stata, K. Bharat, and F. Maghoul. The term vector database: fast access to indexing terms for web pages. *Computer Networks*, 33(1-6):247–255, 2000.

[26] C. T. Yu, K. Lam, and G. Salton. Term weighting in information retrieval using the term precision model. *Journal of the ACM*, 29(1):152–170, 1982.