Search Strategies for Keyword-based Queries

Matthias Hagen and Benno Stein
Bauhaus University Weimar
Germany
<first name>.<last name>@uni-weimar.de

***Abstract*—Given a set of keywords, we find a maximum Web query (containing the most keywords possible) that respects user-defined bounds on the number of returned hits. We assume a real-world setting where the user is not given direct access to a Web search engine's index, i.e., querying is possible only through an interface. The goal to be optimized is the overall number of submitted Web queries.**

One original contribution of our research is the formalization and theoretical foundation of the problem. But, in particular, we develop a co-occurrence probability informed search strategy for the problem. The performance gain achieved with our approach is substantial: compared to the uninformed baseline (without co-occurrence information) the expected savings are up to 20% in the number of submitted queries and runtime.

***Keywords*-Web Search, Query Formulation, Search Cost Optimization, Maximum Query, Long Query Reduction**

I. Introduction

Suppose a typical user of a Web search engine with an information need for which she can come up with a set of potential keywords. The user will submit a query containing some of these words and expects a reply in the form of a ranked result list along with an estimation of the total number of results. Experience shows that very long queries return few or even no hits while rather short Web queries are likely to return millions of documents. Of course, such short queries are often answered reasonably well, i.e., among the top-ranked results the user has a good chance to find a matching item. However, these "good-natured" queries are not the focus of this paper; we address scenarios where the user is not satisfied by the results of her initial query. This case is not uncommon, and search engines provide different means of supporting users in such a situation. Examples include query expansion for queries returning lots of hits or spelling correction for queries returning no hits due to typos. In this paper we present another approach having a more combinatorial flavor, while being easily combinable with existing technology: The *maximum query* for a given set of keywords (a query containing as many of the keywords as possible, while still returning a specified number of results).

The rationale for considering maximum queries is as follows. The number of results a user will consider for a query is usually constrained by a processing capacity $l_{\max}$, determined by the user's reading time etc. If the user faces an underspecific query with millions of hits, she can only check a fraction of the results—typically the top-ranked ones. This puts the burden of selecting the most informative documents on the search engine's ranking algorithm. If the ranking fails to provide relevant documents on top, the user probably formulates a more specific query by including more or different keywords. But what subset of her potential keywords should the user select to be sufficiently descriptive of her information need? We argue that in such situations the most promising queries are the ones that are sufficiently specific to not return millions of hits—but also not just one or two. We suggest to use a maximum query for a set of keywords, i.e., a query being maximally specific (containing as many keywords as possible) while still returning a reasonable number of results (at most $l_{\max}$). The user can then check the entire result list and will not miss any potential match for her information need due to search engine ranking issues that she cannot influence.

Obviously, several queries have to be submitted to identify a maximum query. We solve the problem taking the user perspective, i.e., we give algorithms that are not restricted to be implemented at search engine site. In fact, our algorithms are of *external* nature and just use the standard search engine interfaces. These interfaces usually do not offer direct access to the engine's index and a user has incomplete or even no knowledge of the underlying retrieval model, implementation details, etc. The search engine appears as a black box, acting like an oracle that answers queries. Furthermore, querying is not for free but entails costs—at the very least some non-negligible amount of time is consumed, and monetary charges come into play for larger contingents of queries. We analyze the corresponding economic optimization problem for finding maximum queries:

What (automatic) search strategy minimizes the average number of submitted queries?

The following use cases illustrate that maximum queries can appear in different contexts.

A. Relevance and Applicability of Maximum Queries

Known Item Finding. Assume a user that once had access to a document on the Web but now only can come up with a set of keywords that she thinks occur in the document. Refinding the desired document can be tackled by automatically constructing queries from the user's keywords. These queries should return at least a given number of hits; but also not too many since then the expectation is that the query is not descriptive enough to bring up the known item on the top of the result list. Furthermore, some of the remembered phrases might be wrong and should be omitted. A maximum query provides a way-out.

Dealing with Overspecific Queries. Imagine a user whose Web query did not return any hit when submitted to a Web search engine. Thus, the query is overspecific. Interesting for such a user could be the maximum subset of her query that would still give a reasonable number of hits.

Search Sessions. A search session comprises the set of consecutive Web queries a user submits in order to satisfy her information need. Longer search sessions indicate that the user is not fully satisfied with the results so far. A potential reason is that the user has chosen to query with “bad” combinations of otherwise good keywords. Using a maximum query from all the keywords submitted in the session could help.

B. Related Work

Maximum query formulation is very similar to the task of long query reduction. Long query reduction comprises handling verbose text queries (like description parts of TREC topics or medical search engine queries) but also dealing with keyword queries of more than 4 words. However, all the existing research results on long query reduction [2, 3, 4, 5, 6, 7, 8, 9] assume full access to a search engine’s index and thus do not take into account the user’s costs for querying.

Two papers explicitly deal with the problem of finding queries respecting a bound on the number of returned hits for a given keyword set. Shapiro and Taksa [11] suggest a rather simple open end query formulation approach for which it is straightforward to find situations where the approach fails although appropriate queries exist. A more involved maximal termset method is proposed by Pôssas et al. [10]. However, both approaches focus on finding a whole set of queries instead of just one maximum query and neither Shapiro and Taksa nor Pôssas et al. analyze the number of submitted queries.

II. Notation and Basic Definitions

Starting point of the query formulation process against a Web search engine $\mathcal{S}$ is a set $W = \{w_1, \ldots, w_n\}$ of keywords (it makes no difference to also allow a “keyword” to be a complete phrase). The keywords might be given by a human user or might be automatically generated depending on the use case, e.g., words from the user’s queries combined with automatically derived query expansion terms. Subsets $Q \subseteq W$ can be submitted as Web queries (complete phrases would be included in quotation marks). An engine’s reply to a query Q consists of a constant length head of an exhaustive, ranked list L_Q of snippets and URLs of result documents and an estimation l_Q for the real result list length $|L_Q|$. As for query semantics, we adopt the usual AND notion, i.e., we require every query keyword to be contained in the returned results.

The task of the maximum query problem is to find a query Q having the following properties. First, $l_{\min} \leq l_Q \leq l_{\max}$ for given constant lower and upper bounds $l_{\min}$ and $l_{\max}$. Usually, $l_{\min}$ is set to some small value like 1 or 10 and $l_{\max}$ will be set to the user’s capacity, which typically is at most 100. We say that for $l_Q < l_{\min}$ the query Q is *underflowing*, whereas for

Table I
KEYWORD DOCUMENT RELATIONSHIP IN THE EXAMPLE SCENARIO.

Keyword	**Document**									
	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}
w_1	●	●		●	●					●
w_2		●				●			●	
w_3	●	●	●	●	●	●	●		●	
w_4	●			●		●	●	●		●
w_5	●		●		●	●	●	●	●	

$l_Q > l_{\max}$ it is *overflowing*. Queries that are neither under- nor overflowing are *valid*. A valid query Q is *maximal* iff adding any keyword from $W \setminus Q$ results in an underflowing query. A *maximum* query is a maximal query containing the most keyphrases possible. The corresponding problem is:

MAXIMUM QUERY
Given: (1) A set W of keywords.
(2) A query interface for a Web search engine $\mathcal{S}$.
(3) An upper bound $l_{\max}$ on the result list length.
(4) A lower bound $l_{\min}$ on the result list length.
Task: Find a maximum query $Q \subseteq W$.

In our process of solving MAXIMUM QUERY we count the overall number *cost* of queries that are submitted to $\mathcal{S}$. As for the runtime analysis we are interested in the time t_{Web} consumed by the Web queries and the internal computation time t_{local} for the query formulation process excluding the Web query time. Our assumption is that t_{Web} will clearly dominate t_{local}, i.e., $t_{\text{Web}} \gg t_{\text{local}}$.

To further explain our setting, consider the following example scenario with the ten indexed documents $d_1, \ldots, d_{10}$ and the set $W = \{w_1, \ldots, w_5\}$ of keywords with the keyword document relationship given in Table I. Note that, submitted as a query, the set W itself will not result in any hit on the ten document collection since none of the documents contain all keywords. Let $l_{\min} = 3$ and $l_{\max} = 4$, i.e., we are looking for subsets of the keywords that are contained in at least 3 and at most 4 documents. Figure 1 shows the middle levels of the possible 2^5 queries’ hypercube; valid queries are shown highlighted. An example of an overflowing query is $\{w_3, w_5\}$ (six hits), whereas $\{w_1, w_5\}$ is underflowing (two hits). The family of maximal valid queries $\mathcal{Q}_{\text{up}} = \{\{w_1, w_3\}, \{w_1, w_4\}, \{w_2, w_3\}, \{w_3, w_4, w_5\}\}$ corresponding to Figure 1’s upper border has size 4. Note that $\mathcal{Q}_{\text{up}}$ has a unique maximum element in our example scenario—the maximum query $\{w_3, w_4, w_5\}$.

III. Search Strategies for Maximum Queries

We give two algorithms to solve the MAXIMUM QUERY problem. The first is a baseline algorithm, which is then improved by informing it with a co-occurrence probability graph that is used to internally estimate queries before submission. In both algorithms we use the search engine’s result list length estimations l_Q, although they often overestimate the correct lengths. However, the estimations usually respect monotony (queries containing additional phrases have smaller l-value),

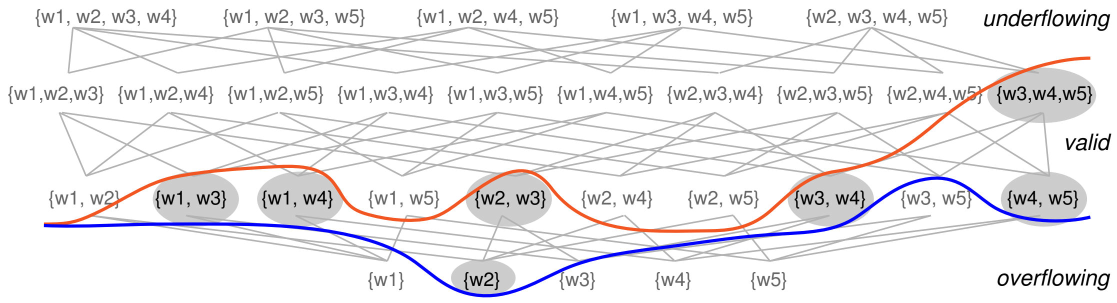

Figure 1. Hypercube of possible queries in the example scenario.

and the shorter the result list, the better the estimations. Hence, in the range of our result constraints (we usually have $l_{\max} = 100$ and thus are looking for maximum queries with at most 100 results) they are quite accurate.

A. The (uninformed) baseline

A pseudocode listing of our baseline method is given as Algorithm 1. It first removes underflowing keywords (lines 1 and 2 of the listing) because they cannot be contained in a maximum query. Note that validity checks (lines 2, 4, 15, and 17) are managed by submitting the query to the engine $\mathcal{S}$. A second pre-check (line 4) ensures that the remaining set of non-underflowing keywords itself is underflowing, since otherwise W itself is maximum or no valid query can be found at all. The main idea of the baseline approach can then be characterized as a depth-first search on a tree containing all possible queries. Revisiting nodes in the tree is prohibited by processing the keywords in the order of their indices. Hence, the algorithm starts trying to find a maximal valid query containing the first keyword w_1. It then adds the keywords w_2, w_3 etc. as long as the query remains non-underflowing. If the query becomes underflowing, the last keyword is removed and the next one tried. If all keywords have been tried and the query is valid, this is the first candidate to be a maximum query. The algorithm now backtracks to other possible paths in the search tree. Pruning is done whenever the algorithm excluded as many keywords as are excluded from the currently stored maximum query. A valid query that is longer than the maximum query so far is stored as the new maximum query. Since this strategy causes an exhaustive search, it is guaranteed to find a maximum query if there is one at all.

Algorithm 1 Baseline algorithm for MAXIMUM QUERY

```
    Input: $W = \{w_1, \dots, w_n\}$, $l_{\min}$, and $l_{\max}$
    Output: a maximum valid query $Q_{\max} \subseteq W$

 1: for all $w \in W$ do
 2:     if $\{w\}$ is underflowing then $W \leftarrow W \setminus \{w\}$
 3: $Q_{\max} \leftarrow \emptyset$
 4: if $W$ is underflowing then
 5:     while $(W \neq \emptyset) \wedge (|W| > |Q_{\max}|)$ do
 6:         $w \leftarrow$ keyword with lowest index from $W$
 7:         $W \leftarrow W \setminus \{w\}$
 8:         ENLARGE($\{w\}$, $W$)
 9:     output $Q_{\max}$
10: else output $\{W\}$

11: procedure ENLARGE(query $Q$, keywords $W_{\text{left}}$)
12:     while $(W_{\text{left}} \neq \emptyset) \wedge (|Q| + |W_{\text{left}}| > |Q_{\max}|)$ do
13:         $w \leftarrow$ keyword with lowest index from $W_{\text{left}}$
14:         $W_{\text{left}} \leftarrow W_{\text{left}} \setminus \{w\}$
15:         if $Q \cup \{w\}$ is overflowing or valid then
16:             $Q' \leftarrow$ ENLARGE($Q \cup \{w\}$, $W_{\text{left}}$)
17:             if $Q'$ is valid and $|Q'| > |Q_{\max}|$ then
18:                 $Q_{\max} \leftarrow Q'$
19:     return $Q$
```

B. Co-occurrence informed improvement

A drawback of the described (uninformed) baseline is that it submits every intermediate query candidate to the search engine. To decrease the number of submitted Web queries, we improve the baseline by informing it with the keywords' co-occurrence probabilities. A pre-processing step of the improved version initializes a vertex and edge weighted directed co-occurrence graph G_W, storing as weights the l-values and co-occurrence probabilities of the keywords. In our first experiments we also submitted Web queries to derive the weights in the graph but did not count them for the overall cost. The rationale is that in case of substantial savings achievable by using the graph, a very promising future research task is initializing the graph with a local "sandbox" corpus on which co-occurrence probability computation can be done at zero cost (e.g., a local index of Wikipedia documents). In this paper we show the potential of the co-occurrence graph technique and, thus, describe initializing G_W using Web queries. The graph contains a vertex v_w for each keyword $w \in W$. The weight of v_w is set to $l_{\{w\}}$. We have two edges connecting vertices v_w and $v_{w'}$. An edge $e = v_w \rightarrow v_{w'}$ from v_w to $v_{w'}$ gets as weight the yield factor $\gamma(e) = l_{\{w,w'\}}/l_{\{w\}}$. This factor multiplied by the weight of v_w gives the yield of Web hits when w' is added to w. Note that the yield factor is reminiscent of the co-occurrence probability for the keywords w and w'. The graph G_W itself is reminiscent of a mutual information graph. Figure 2 shows the co-occurrence graph for the example scenario from Table I.

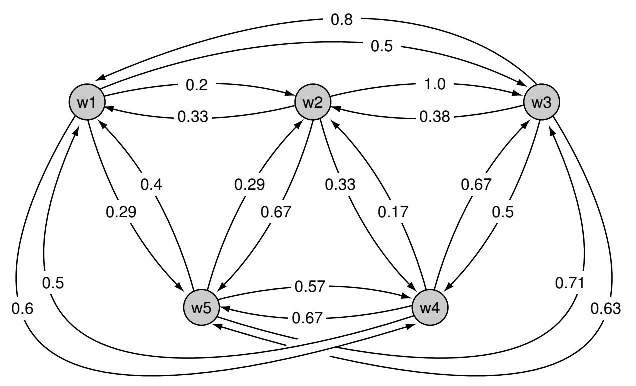

Figure 2. The co-occurrence graph for the example scenario from Table I.

Using G_W the algorithm can internally estimate a query candidate; and only submit it as a Web query if really necessary. Assume the current examined query is Q_{cand}. Moreover, assume that all queries Q from previous computation steps have a stored value est_Q indicating an estimation of the length of their result lists. Before submitting Q_{cand} as a Web query, $est_{Q_{\text{cand}}}$ is internally estimated as follows. Let w' be the last added keyword. Hence, we already have the value $est_{Q'}$ for the query $Q' = Q_{\text{cand}} \setminus \{w'\}$. The algorithm now sets $est_{Q_{\text{cand}}} = est_{Q'} \cdot \text{avg}\{\gamma(v_w \to v_{w'}) : w \in Q'\}$, where avg denotes the mean value. Submitting Q_{cand} as a Web query and storing the engine's $l_{Q_{\text{cand}}}$ as $est_{Q_{\text{cand}}}$ is done iff $est_{Q_{\text{cand}}} < 5 \cdot l_{\max}$. The adjustment factor 5 in the last inequality was determined experimentally. If $est_{Q_{\text{cand}}} \geq 5 \cdot l_{\max}$ the algorithm does not submit a Web query but stores the internally derived $est_{Q_{\text{cand}}}$. This reduces the number of submitted Web queries. Furthermore, experiments showed that usually $l_Q \geq est_Q$ such that the co-occurrence informed approach usually does not "miss" valid queries the uninformed baseline finds.

IV. Experimental Analysis

We experimentally compare the co-occurrence informed search strategy to the uninformed baseline on the known item finding use case (cf. Section I-A). The setup is as follows: from a given document, we extract a set of keywords and then run both algorithms on it. Keyword extraction is managed by an implementation of the head noun extractor described in [1]. Our document collection was obtained by crawling papers on computer science from major conferences and journals. From the established corpus we removed the documents for which we were not able to extract 15 reasonable keywords. Our test collection was formed by the 775 remaining documents. We set the bounds $l_{\max} = 100$ and $l_{\min} = 10$. For each document of the test collection we had runs of the algorithms with $3, 4, \ldots, 15$ extracted keywords. Note that we cannot expect savings for 1 or 2 keywords. As a Web search engine we used the Microsoft Bing API.

The results of our experiments can be found in Table II. The first row states the number of processed documents from which the respective keywords are extracted. Especially for sets with few keywords, a maximum query often could not be found as the complete query containing all keywords is overflowing (cf. second row). We filter out such keyword sets and derive the statistics (rows 4 to 14) just for the remaining documents (number given in third row). In these cases both algorithms always found a maximum query and the keyword extraction source document always was among the search engine's returned results. Hence, maximum queries can be a reasonable tool to handle the known item finding use case.

In rows 4 and 5 we state the average number *cost* of Web queries needed to solve Maximum Query with the informed search approach and the uninformed baseline. The average ratio of submitted queries of the informed approach vs. the uninformed baseline is given in row 6 (row 4 divided by row 5). A visualization of the ratios' behavior is given in Figure 3. The possible savings using a co-occurrence informed search are substantial; up to 20% of the queries compared to the uninformed baseline. Surprisingly, there is a huge "drop" in the possible savings for 15 keywords as well as a huge increase in the number of submitted queries for the informed method. The reason probably is the following. The 15-th extracted keyword w_{15} often is not as descriptive of the document's topic as the previous 14 are. (Note that for each document the first 14 extracted keywords in the 15 keywords case are identical to the extracted keywords in the 14 keywords case). The "non-descriptiveness" of w_{15} causes w_{15}'s yield factors to be very small. Thus, many of the queries containing w_{15} have an internal estimation below the informed search's $(5 \cdot l_{\max})$-bound such that these queries are submitted and increase the overall cost.

The time consumption statistics in rows 7 to 11 show the expected behavior: The internal computation time t_{local} is orders of magnitude lower than t_{Web} for both approaches.

Finally, in rows 12 to 14 we report statistics on the average size and "quality" of the generated queries. The slightly smaller maximum queries for the informed approach are due to some rare internal overestimations, i.e., queries with $est_Q > l_Q$. Such overestimations potentially "hide" some maximum queries from the informed approach but the uninformed baseline finds them. This also results in slightly different result sets: the ratio of URLs the informed approach's maximum query and the uninformed baseline's maximum query have in common vs. the URLs of the uninformed

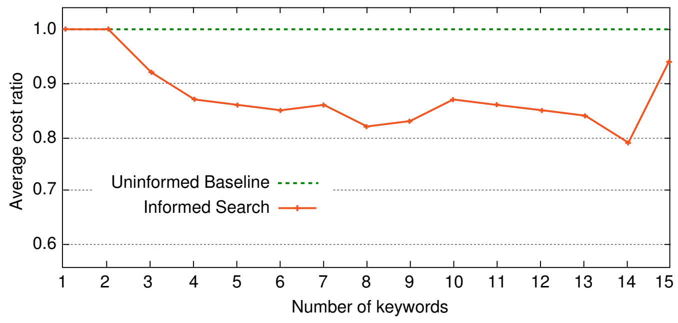

Figure 3. Average cost ratio over baseline to find a maximum query.

Table II
EXPERIMENTAL RESULTS.

		Number of keywords												
		3	4	5	6	7	8	9	10	11	12	13	14	15
1	Processed documents	775	775	775	775	775	775	775	775	775	775	775	775	775
2	No maximum query possible	711	650	595	540	479	402	337	328	218	153	115	99	86
3	Maximum query found	64	125	180	235	296	373	438	447	557	622	660	676	689
4	Avg. *cost* informed	6.09	8.49	10.90	13.21	16.23	20.32	24.44	27.01	37.30	48.58	73.38	71.14	108.78
5	Avg. *cost* uninformed	6.61	9.70	12.67	15.61	18.94	24.94	29.40	30.94	43.55	56.97	87.37	90.17	116.22
6	**Avg. cost ratio informed vs. uninformed**	**0.92**	**0.87**	**0.86**	**0.85**	**0.86**	**0.82**	**0.83**	**0.87**	**0.86**	**0.85**	**0.84**	**0.79**	**0.94**
7	Avg. t_{local} informed (ms)	0.45	0.69	0.95	1.05	1.49	1.89	2.85	2.90	4.41	6.04	9.31	8.67	14.44
8	Avg. t_{Web} informed (s)	1.57	2.18	2.75	3.41	4.14	4.97	6.91	9.11	11.01	15.28	24.42	26.18	44.04
9	Avg. t_{local} uninformed (ms)	1.55	1.48	2.08	2.42	2.92	3.66	4.83	4.83	6.33	8.17	12.36	12.76	25.06
10	Avg. t_{Web} uninformed (s)	1.70	2.49	3.20	4.03	4.83	6.10	8.31	10.43	12.85	17.92	29.08	33.18	47.06
11	Avg. Web query time (ms)	257.96	256.64	252.28	258.01	254.82	244.52	282.63	337.09	295.07	314.61	332.77	367.98	404.86
12	Avg. size maximum query informed	1.25	2.18	3.21	4.02	4.99	5.84	6.77	7.83	8.37	8.96	9.50	10.14	10.55
13	Avg. size maximum query uninformed	1.31	2.18	3.21	4.03	5.02	5.88	6.82	7.90	8.41	8.98	9.52	10.15	10.57
14	Avg. ratio of common result URLs	0.95	0.99	0.99	0.98	0.96	0.97	0.91	0.94	0.94	0.96	0.94	0.93	0.95

baseline is given in line 14. Note that on average the informed strategy's maximum query only misses at most 10% of the uninformed baseline's URLs. This difference is rather small and intensive spot checks showed that usually both approaches found the same maximum query (and thus the same result set).

V. CONCLUSION AND OUTLOOK

We showed the need for a user-oriented query cost analysis in the process of formulating maximum queries against a Web search engine. In such situations a user "plays" against the engine in order to satisfy her information need by submitting keyword queries. Our formalization forms the ground for both to define the according problem MAXIMUM QUERY and to develop search strategies to solve it. The strategies are applicable in numerous situations and the co-occurrence informed approach should be used instead of the uninformed baseline method as is experimentally underpinned by the substantial savings in the number of submitted queries.

Note that our approaches can also be implemented as a background process at search engine site. Maximum query suggestions then could have a potential of improving user experience in case of unsuccessful search sessions or queries with empty result lists.

An interesting task for future work is to analyze the use of potential "sandboxes" from which the co-occurrence probabilities can be derived at zero cost. Another important issue is to further analyze and trim the adjustment factor that helps to decide when to submit a query to the search engine. This might help to overcome the observed behavior in the 15 keywords case.

REFERENCES

[1] Ken Barker and Nadia Cornacchia. Using noun phrase heads to extract document keyphrases. In *Proceedings of AI 2000*, pages 40–52.

[2] Michael Bendersky and W. Bruce Croft. Discovering key concepts in verbose queries. In *Proceedings of SIGIR 2008*, pages 491–498.

[3] Giridhar Kumaran. *Interactive Reformulation of Long Queries*. PhD thesis, Graduate School of the University of Massachusetts Amherst, May 2008.

[4] Giridhar Kumaran and James Allan. A case for shorter queries, and helping users create them. In *Proceedings of HLT 2007*, pages 220–227.

[5] Giridhar Kumaran and James Allan. Adapting information retrieval systems to user queries. *Information Processing and Management*, 44(6): 1838–1862, 2008.

[6] Giridhar Kumaran and James Allan. Effective and efficient user interaction for long queries. In *Proceedings of SIGIR 2008*, pages 11–18.

[7] Giridhar Kumaran and Vitor R. Carvalho. Reducing long queries using query quality predictors. In *Proceedings of SIGIR 2009*, pages 564–571.

[8] Matthew Lease, James Allan, and W. Bruce Croft. Regression rank: Learning to meet the opportunity of descriptive queries. In *Proceedings of ECIR 2009*, pages 90–101.

[9] Gang Luo, Chunqiang Tang, Hao Yang, and Xing Wei. MedSearch: a specialized search engine for medical information retrieval. In *Proceedings of CIKM 2008*, pages 143–152.

[10] Bruno Pôssas, Nivio Ziviani, Berthier A. Ribeiro-Neto, and Wagner Meira Jr. Maximal termsets as a query structuring mechanism. In *Proceedings of CIKM 2005*, pages 287–288.

[11] Jacob Shapiro and Isak Taksa. Constructing web search queries from the user's information need expressed in a natural language. In *Proceedings of SAC 2003*, pages 1157–1162.