2010 IEEE/WIC/ACM International Conference on
Web Intelligence and Intelligent Agent Technology
ISBN 978-0-7695-4191-4, pp. 90-97, ©IEEE 2010.

Making the Most of a Web Search Session

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

***Abstract*—We tackle problems related to Web query formulation: given the set of keywords from a search session, 1) we find a maximum promising Web query, and, 2) we construct a family of promising Web queries covering all keywords. A query is promising if it fulfills user-defined constraints on the number of returned hits. We assume a real-world setting where the user is not given direct access to a 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.**

For both problems we develop search strategies based on co-occurrence probabilities. The achieved performance gain is substantial: compared to the uninformed baselines without co-occurrence probabilities the expected savings are up to 50% in the number of submitted queries, index accesses, and runtime.

***Keywords*-Web Search Session, Query Formulation, Query Cost Optimization, Maximum Query, Query Cover**

I. Introduction

The interactions between Web search users and search engines follow a classic scheme. The user comes up with a set of (in her opinion) appropriate keywords for a given information need. She submits a query containing some of these keywords and gets back a ranked result list. If the user does not find a match for her information need among the first results, she will hardly browse all the items but submit different queries based on her keywords until she is satisfied or decides to give up. This process forms a *search session*—the set of consecutive Web queries a user submits to a search engine in order to satisfy a given information need.

Experience shows that in many cases a user's first try is answered reasonably well, i.e., the first query brings up an appropriate result. Such one-query-sessions are not the focus of this paper; we consider longer sessions where the user did not succeed with her first query. Search engines provide different means to support unsuccessful users, e.g., query expansion for queries returning lots of hits or spelling correction for queries returning no hits due to typos. In this paper we present two other approaches having a more combinatorial flavor, while being easily combinable with existing technology:

1) The *maximum query* for a given set of keywords, i.e., a query containing as many of the keywords as possible, while returning a reasonable number of results.
2) The *query cover* for a given set of keywords, i.e., a family of queries (each returning a reasonable number of results) that contain as many keywords as possible.

The idea is to use the keywords submitted in a search session up to the current query, and suggest a maximum query or a query cover as the user's next query. The requirement to contain as many of the keywords as possible reflects the following rationale. Taken together, the keywords of a search session describe the user's information need. Some of the keywords might not be appropriate (e.g. typos) and should be omitted, but the more keywords are contained in a maximum query or a query cover the better is the descriptiveness of the user's information need.

The rationale for requiring a reasonable number of hits per query also deserves closer consideration. Queries with empty result pages are useless and the same often applies to queries returning only a handful of hits. This gives a lower bound on the number of desired results. But there is also an upper bound since the number of results a user will consider for a single query is usually constrained by a processing capacity $l_{\max}$, determined by the user's reading time etc. If the user faces a query with millions of hits, she can only check a fraction of the results—typically the top-ranked ones. Relevant entries below are missed. We argue that the best queries are the ones that are sufficiently specific to not return millions of hits—but also not just one or two. For such queries the user can check the complete result list and will not miss any potential match for her information need due to search engine ranking issues that she cannot influence.

Hence, from the user's perspective, maximum and covering queries contain the best possible description of the information need and offer the chance to check all the results. However, finding a maximum query or a query cover "by hand" is not that straightforward. Several queries have to be submitted to identify appropriate keyword combinations. Hardly any user will take the time for such a lengthy procedure. Therefore, we give algorithms for both tasks. To be applicable at user site the algorithms are of *external* nature, i.e., they only use standard search engine interfaces. The Web search engine is handled as a black box, acting like an oracle that answers queries. There is no need to know the underlying retrieval model or implementation details.

Since Web searches are not for free but entail 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 and covering queries. We ask:

Which strategy minimizes
the average number of submitted queries?

A. Related Work

A lot of research has been done on approaches for better results on better queries. An example of a very promising

idea is to estimate or to predict a given query's performance [5, 6, 7, 9, 10, 11, 12, 22]. Query quality *estimators* have a "post-retrieval nature," using knowledge on the already processed query and the retrieved documents. But there is also a lot of effort in analyzing pre-retrieval *predictors*, which can be evaluated *before* processing the query. Especially these pre-retrieval predictors could be interesting for avoiding the submission of too many queries. However, the evaluation of most predictors needs access to knowledge a standard Web search interface does not provide. For example, the simplified query clarity predictor [12] needs the total keyword frequencies for the whole corpus, but Web search engines just return an estimate of the number of documents in the corpus that contain the keyword. The query scope predictor [12] needs the number of documents in the index, but most Web search providers stopped publishing it. The mutual information based predictors [14, 17] need the frequency of two keywords in a sliding window with given size over the whole corpus, but no engine reports such values.

Nevertheless, two long query reduction methods successfully use query quality predictors [17, 19]—but with unrestricted access to a search engine's index. The task of *long query reduction* comprises handling queries with more than 4 keywords and processing verbose text queries (like the description parts of TREC topics or queries to medical search engines). The interest in the issue of how to handle long queries is on the rise [2, 13, 14, 15, 16, 18], as a typical Web query nowadays is becoming longer and longer or "more verbose." Our setting can be seen as an instance of long query reduction since all the keywords from a search session can be viewed as a single long query. Note, however, that the existing research on long query reduction assumes full access to a search engine's index and that it is not user-oriented in the sense that the costs for querying are not taken into account.

Two papers explicitly deal with the problem of formulating queries from a given set of keywords respecting a bound on the number of returned hits [20, 21]. Shapiro and Taksa [21] 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 method for query formulation is proposed by Pôssas et al. [20], the so-called maximal termset query formulation. However, neither Shapiro and Taksa nor Pôssas et al. analyze the number of submitted queries.

B. Paper Organization

All previous results on problems related to finding maximum or covering queries can be characterized as being system-oriented: they disregard the costs of problem solving at user site. Therefore, in Section II, we carefully introduce formal notions for user-oriented problem definitions and show the effect in an example scenario.

From the baseline algorithms solving the problems (cf. Section III) we develop informed search strategies, equipping the algorithms with a co-occurrence probability graph (cf. Section IV). The experimental analysis in Section V demonstrates that substantial improvements with respect to the number of submitted queries are possible. Depending on the problem, our informed approaches with co-occurrence graph save up to 50% of Web queries on average. Our investigations close with concluding remarks and an outlook on future work in Section VI.

II. Notation, Basic Definitions, and an Example Scenario

We formally describe our query formulation framework from the user perspective against a Web search engine $\mathcal{S}$. Starting point is a set $W = \{w_1, \ldots, w_n\}$ of keywords. Note that it makes no difference to 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 1) a constant length head of a ranked list L_Q of snippets and URLs of documents containing the keywords from Q, and 2) an estimation l_Q for the real result list length $|L_Q|$. We adopt the usual AND notion for Web queries, i.e., every query keyword has to be contained in every result.

The task of the query cover problem is to find a family $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$ of queries $Q_i \subseteq W$ having the following properties. First, $\mathcal{Q}$ should be *simple* in the sense that $Q_i \not\subseteq Q_j$ for any $Q_i, Q_j \in \mathcal{Q}$, with $i \neq j$. This avoids redundancy in the queries. Second, a query $Q \in \mathcal{Q}$ has to satisfy $l_{\min} \leq l_Q \leq l_{\max}$ 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. Adopting the notation from [1] we say that for $l_Q < l_{\min}$ the query Q is *underflowing*, whereas for $l_Q > l_{\max}$ it is *overflowing*. Queries that are neither under- nor overflowing are *valid*. A valid query Q is *minimal* iff leaving out any keyword from Q would result in an overflowing query; it is *maximal* iff adding any keyword from $W \setminus Q$ would result in an underflowing query.

A query Q *covers* all the keywords in Q. Analogously, a family $\mathcal{Q}$ of queries covers all the keywords contained in $\bigcup_{Q \in \mathcal{Q}} Q$. Note that there are situations where it is not possible to cover W with a family of valid queries, e.g., when a single keyword itself is underflowing. A keyword $w \in W$ is *coverable* iff there is a valid query $Q \subseteq W$ with $w \in Q$. We can formally state the query cover problem as follows.

Query Cover

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 simple family $\mathcal{Q} \subseteq 2^W$ of valid queries covering the coverable keywords from W.

In our process of finding the output $\mathcal{Q}$ on input W we count the overall number *cost* of queries that are submitted

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	•		•		•	•	•	•	•	

to $\mathcal{S}$. As for the analysis of the runtime of our system 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. We have the 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 hypercube of the possible 2^5 queries; 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). We have $|\mathcal{Q}| \leq 5$ since we require $\mathcal{Q}$ to be simple. A family reaching this bound is $\mathcal{Q}_{\text{lo}} = \{\{w_1, w_3\}, \{w_1, w_4\}, \{w_2\}, \{w_3, w_4\}, \{w_4, w_5\}\}$—the with respect to set inclusion minimal valid queries—corresponding to the lower border in Figure 1. 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 only.

There are many possible solutions to QUERY COVER in our example scenario, e.g., besides $\mathcal{Q}_{\text{lo}}$ or $\mathcal{Q}_{\text{up}}$ we could give the family $\mathcal{Q} = \{\{w_1, w_3\}, \{w_2\}, \{w_3, w_4, w_5\}\}$. However, $\mathcal{Q}$ is a mix of queries from $\mathcal{Q}_{\text{lo}}$ and $\mathcal{Q}_{\text{up}}$. A user that further demands the solution to be close to $\mathcal{Q}_{\text{up}}$ or close to $\mathcal{Q}_{\text{lo}}$ in order to get longer or shorter queries covering her keywords (and thus expecting fewer or more results in the final result lists) would not be satisfied with such a mixture.

We can show that $\mathcal{Q}_{\text{lo}}$ and $\mathcal{Q}_{\text{up}}$ always cover the coverable keywords of W but first have to prove the following Lemma.

Lemma 1: Let $\mathcal{Q}$ be a family of valid queries covering the coverable keywords from a set W. For every $Q \in \mathcal{Q}$ we have: (i) there is a sub-family $\mathcal{Q}'_{\text{lo}} \subseteq \mathcal{Q}_{\text{lo}}$ such that $Q = \bigcup_{Q' \in \mathcal{Q}'_{\text{lo}}} Q'$ and (ii) there is some $Q_{\text{up}} \in \mathcal{Q}_{\text{up}}$ such that $Q \subseteq Q_{\text{up}}$.

Proof: (i) Assume we have $Q \in \mathcal{Q}$ but $Q \neq \bigcup_{Q' \in \mathcal{Q}'_{\text{lo}}} Q'$ for any $\mathcal{Q}'_{\text{lo}} \subseteq \mathcal{Q}_{\text{lo}}$. Since $\mathcal{Q}$ is a family of valid queries, Q must be valid. Assume Q just contains coverable keywords from W. Now consider the family $\mathcal{Q}'$ of the $2^{|Q|}-1$ subqueries of Q excluding the empty query. Let $\mathcal{Q}'' \subseteq \mathcal{Q}'$ be the sub-family of valid queries. Note that $\mathcal{Q}''$ is not empty since it contains Q. From $\mathcal{Q}''$ we remove all queries that are proper supersets of queries in $\mathcal{Q}''$ and get the family $\tilde{\mathcal{Q}}$ of minimal valid subqueries of Q. Note that $\tilde{\mathcal{Q}}$ is not empty since $\mathcal{Q}''$ is not and that $Q = \bigcup_{\tilde{Q} \in \tilde{\mathcal{Q}}} \tilde{Q}$. But since $\tilde{\mathcal{Q}}$ contains minimal valid queries only, we have $\tilde{\mathcal{Q}} \subseteq \mathcal{Q}_{\text{lo}}$; a contradiction to our assumption. Hence, Q contains a non-coverable keyword $w \in W$. But since w is not coverable by a valid query, Q cannot be valid. A contradiction again.

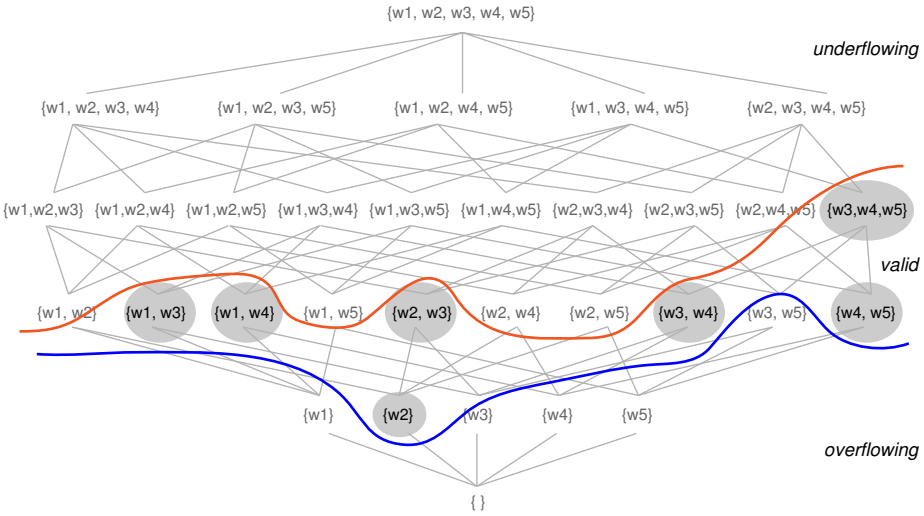

Figure 1. Hypercube of possible queries in the example scenario.

(ii) Assume we have $Q \in \mathcal{Q}$ but $Q \not\subseteq Q_{\mathrm{up}}$ for any $Q_{\mathrm{up}} \in \mathcal{Q}_{\mathrm{up}}$. Since $\mathcal{Q}$ is a family of valid queries, Q must be valid. Assume Q just contains coverable keywords from W. Then there obviously is a maximal superset $Q' \supseteq Q$ being valid (it can be found by adding as many of the keywords from $W \setminus Q$ as possible). But since $Q' \in \mathcal{Q}_{\mathrm{up}}$ we have a contradiction. Hence, Q contains a non-coverable keyword $w \in W$. But since w is not coverable by a valid query, Q cannot be valid. A contradiction again. ∎

A direct consequence of Lemma 1 and the definitions of $\mathcal{Q}_{\mathrm{lo}}$ and $\mathcal{Q}_{\mathrm{up}}$ is the following "cover-ability" of $\mathcal{Q}_{\mathrm{lo}}$ and $\mathcal{Q}_{\mathrm{up}}$.

Corollary 1: For a given set W of keywords, the families $\mathcal{Q}_{\mathrm{lo}}$ and $\mathcal{Q}_{\mathrm{up}}$ cover the coverable keywords. Furthermore, $\mathcal{Q}_{\mathrm{lo}}$ contains the with respect to set inclusion minimal queries covering the coverable keywords, whereas $\mathcal{Q}_{\mathrm{up}}$ contains the maximal queries.

However, note that usually not all queries from $\mathcal{Q}_{\mathrm{lo}}$ or all from $\mathcal{Q}_{\mathrm{up}}$ are needed to cover the coverable keywords, e.g., in our example scenario. Hence, we consider the following two QUERY COVER variants that both favor query covers from which no query can be left out without violating coverage of the coverable keywords.

MINIMAL QUERY COVER

Given: A QUERY COVER instance.
Task: Find a family $\mathcal{Q} \subseteq \mathcal{Q}_{\mathrm{lo}}$ containing as few queries as possible that covers the coverable keywords from W.

MAXIMAL QUERY COVER

Given: A QUERY COVER instance.
Task: Find a family $\mathcal{Q} \subseteq \mathcal{Q}_{\mathrm{up}}$ containing as few queries as possible that covers the coverable keywords from W.

As for our example scenario from Table I, note that the query family $\{\{w_1, w_3\}, \{w_2\}, \{w_3, w_4, w_5\}\}$ solves the original QUERY COVER problem but not MINIMAL nor MAXIMAL QUERY COVER. Instead, a possible solution for MINIMAL QUERY COVER is $\{\{w_1, w_3\}, \{w_2\}, \{w_4, w_5\}\}$ while the family $\{\{w_1, w_3\}, \{w_2, w_3\}, \{w_3, w_4, w_5\}\}$ solves MAXIMAL QUERY COVER. From Lemma 1 and the definition of $\mathcal{Q}_{\mathrm{up}}$ it is straightforward to see that a solution to MAXIMAL QUERY COVER always is a smallest possible QUERY COVER solution.

Besides a cover of her keywords, a user might also be interested in the maximum valid query possible; the respective problem is defined as follows.

MAXIMUM VALID QUERY

Given: A QUERY COVER instance.
Task: Find a valid query $Q \subseteq W$ containing the most keywords possible.

In our example scenario from Table I we have the unique solution $\{w_3, w_4, w_5\}$ for MAXIMUM VALID QUERY.

Note that the decision versions of MINIMAL and MAXIMAL QUERY COVER, as well as MAXIMUM VALID QUERY are NP-complete problems (possible reductions from SET COVER and MAXIMAL INDEPENDENT SET). However, the number of keywords a real search engine accepts usually is bounded. Thus, the problem instances found in practice can be solved by our algorithms with reasonable runtime (cf. the next sections).

III. BASELINE ALGORITHMS

In this section we develop baseline algorithms for the above defined problems MINIMAL and MAXIMAL QUERY COVER, and MAXIMUM VALID QUERY. In these algorithms we use the Web search engine's estimations on the result list lengths (the l-values in our setting), although current Web search engines often overestimate the correct result list lengths in practice. However, the engines' estimations usually respect monotony (queries containing additional keywords have smaller l-value), and the shorter the result list, the better the estimations. Hence, in the range of our user constraints—where we usually require at most 100 hits—they are quite accurate.

A. Query cover computation

A straightforward approach for solving MINIMAL and MAXIMAL QUERY COVER would be to compute $\mathcal{Q}_{\mathrm{lo}}$ and $\mathcal{Q}_{\mathrm{up}}$ and then, in a brute-force manner, try to drop some of the queries. However, our experimental pre-tests on our testbed (cf. Section V) revealed that computation of $\mathcal{Q}_{\mathrm{lo}}$ and $\mathcal{Q}_{\mathrm{up}}$ becomes very expensive: for $|W| = 10$, we need about 1400 Web queries on average for exactly computing $\mathcal{Q}_{\mathrm{lo}}$ and more than 650 queries for an approximate solution. Thus, we develop baseline algorithms for the covering problems that avoid computing $\mathcal{Q}_{\mathrm{lo}}$ or $\mathcal{Q}_{\mathrm{up}}$. These baselines can be characterized as greedy backtracking algorithms. A respective pseudo-code listing for the MINIMAL QUERY COVER problem is given as Algorithm 1.

Lines 1 to 4 contain pre-checks to avoid unnecessary computations, e.g., removing underflowing keywords as they cannot be contained in any valid query. Afterwards, a first valid query Q is tried to be constructed containing the keyword w_1 (line 7 of Algorithm 1). As long as the query remains overflowing, the next keywords $w_2, w_3, \ldots$ are added (procedure ENLARGE). Determining a query's validity (lines 2, 4, 18, and 21) causes its submission to the Web search engine $\mathcal{S}$. If the intermediate query becomes underflowing, the last added keyword is removed and the next one tried in a backtracking manner. Backtracking stops whenever the query becomes valid, which is the "greedy" part of the strategy. The query is then added to the output $\mathcal{Q}$ (line 10; note that in the listing of Algorithm 1 the family $\mathcal{Q}$ is intended to be a global variable). The next query is constructed starting from the keyword with the lowest index that is not already covered (line 7). If for an intermediate query all of the not yet covered keywords have been tried and the query still overflows, the algorithm once tries to add keywords from the set of already covered keywords (line 22). This might lead to queries that are supersets of queries already contained in $\mathcal{Q}$. Hence, before outputting the final $\mathcal{Q}$, the procedure SIMPLIFY removes subqueries of larger queries.

Algorithm 1 Greedy algorithm for MINIMAL QUERY COVER

Input: Set $W = \{w_1, \ldots, w_n\}$ of keywords, $l_{\min}$, and $l_{\max}$
Output: Family $\mathcal{Q}$ of minimal valid queries covering W

1: **for all** $w \in W$ **do**
2: **if** $\{w\}$ is underflowing or valid **then** $W \leftarrow W \setminus \{w\}$
3: **if** $\{w\}$ is valid **then** $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\{w\}\}$
4: **if** W is underflowing **then**
5: $W_{uncov} \leftarrow W$
6: **while** $W_{uncov} \neq \emptyset$ **do**
7: $w \leftarrow$ keyword with lowest index from W_{uncov}
8: $Q \leftarrow$ ENLARGE($\{w\}$, $W_{uncov} \setminus \{w\}$, `true`)
9: **if** Q is valid **then**
10: $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{Q\}$
11: $W_{uncov} \leftarrow W_{uncov} \setminus Q$
12: **output** SIMPLIFY($\mathcal{Q}$)
13: **else output** $\mathcal{Q} \cup \{W\}$
14: **procedure** ENLARGE(query Q, keywords W, Boolean cov)
15: **while** $W \neq \emptyset$ **do**
16: $w \leftarrow$ keyword with lowest index from W
17: $W \leftarrow W \setminus \{w\}$
18: **if** $Q \cup \{w\}$ is valid **then return** $Q \cup \{w\}$
19: **if** $Q \cup \{w\}$ is overflowing **then**
20: $Q' \leftarrow$ ENLARGE($Q \cup \{w\}$, W, cov)
21: **if** Q' is valid **then return** Q'
22: **if** cov **then return** ENLARGE(Q, $\bigcup_{Q \in \mathcal{Q}} Q$, `false`)
23: **procedure** SIMPLIFY(query family $\mathcal{Q}$)
24: **for all** $Q \in \mathcal{Q}$ **do**
25: **for all** $Q' \in \mathcal{Q}, Q' \neq Q$ **do**
26: **if** $Q' \subseteq Q$ **then** $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{Q'\}$

The greedy algorithm for MAXIMAL QUERY COVER works analogously. The only difference is in the addition of found valid queries to $\mathcal{Q}$. The MAXIMAL QUERY COVER algorithm does not stop when the current candidate becomes valid but tries to add further uncovered keywords instead. Hence, the procedure SIMPLIFY is not needed anymore.

Note that whenever a query Q becomes valid for the first time in case of solving MINIMAL QUERY COVER, we immediately add it to $\mathcal{Q}$ and do not try to find smaller valid subqueries of Q. Analogously, in case of solving MAXIMAL QUERY COVER, we immediately add a query to $\mathcal{Q}$ if it is valid and cannot be enlarged by any uncovered keyword. This is due to the greedy characteristic of our algorithms and it is supported by the following facts: Let w be the currently explored keyword (line 7 of Algorithm 1). For a given valid query Q, finding a subset $Q' \subset Q$ from $\mathcal{Q}_{\text{lo}}$ containing w (otherwise we cannot guarantee to cover w) needs another $|Q| - 1$ Web queries (for each of the keywords from $Q \setminus \{w\}$, try to exclude it and still have a valid query); finding a superset $Q' \supset Q$ from $\mathcal{Q}_{\text{up}}$ requires another $|W \setminus Q|$ Web queries (for each of the keywords from $W \setminus Q$, try to include it and still have a valid query). We assume such a process to be too costly for our algorithms. Thus, due to these practicability reasons, we relax the constraints of the covering problems and are satisfied by queries close to $\mathcal{Q}_{\text{lo}}$ or close to $\mathcal{Q}_{\text{up}}$ instead of real membership. The output families $\mathcal{Q}$ will still cover the coverable keywords but the contained queries do not have to be contained in $\mathcal{Q}_{\text{lo}}$ or $\mathcal{Q}_{\text{up}}$.

B. Maximum queries

As for the MAXIMUM VALID QUERY problem, we use a depth-first search (similar to Algorithm 1) on a search tree containing all possible queries. Again, 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 maximum valid query containing the first keyword w_1. It adds keywords $w_2, w_3, \ldots$ as long as the query remains non-underflowing. If the query becomes underflowing, the last added keyword is removed and the next one tried. If all keywords have been tried and the query is valid, this is the current candidate to be a maximum query. And now, instead of directly going for a completely new query starting with another keyword (as Algorithm 1 does), the MAXIMUM VALID QUERY algorithm backtracks to other possible queries containing w_1. Pruning is done whenever the algorithm excluded as many keywords as are excluded from the currently stored maximum query. In case of more than one query of maximum size this pruning strategy ensures that the algorithm outputs the lexicographically first maximum query with respect to the initial keyword ordering. Thus, keywords with lower index implicitly are more important. This is a reasonable assumption as it reflects the idea that users in their queries first type the words that are most descriptive of their information need. If in the depth-first search a valid query is found that is longer than the maximum query so far, it is stored as the new maximum query. The described algorithm is guaranteed to find a maximum query—if there is one at all—as it implements an exhaustive search.

IV. IMPROVED INFORMED SEARCH STRATEGIES

To decrease the number of submitted Web queries, we inform the baseline algorithms described in Section III with a pre-processing step that 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 from W. 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 at user site on which co-occurrence probability computation can be done at zero cost (e.g., a local index of Wikipedia documents). In case of implementation at search engine site co-occurrence informations would also be pre-processed and come without any cost. 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'. Like the similarity measures in [3, 4] we obtain the yield factors by

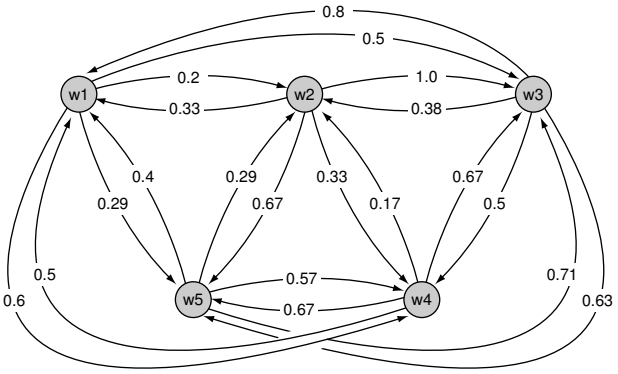

Figure 2. The co-occurrence graph of the example scenario from Table I.

Web queries. The graph G_W itself is reminiscent of a mutual information graph. In Figure 2 we show the co-occurrence graph of our example scenario from Table I.

Using G_W the algorithms can internally estimate a query and only submit it to the engine if really necessary. Let Q_{cand} be the currently examined query and assume that every query Q from a previous computation step has a stored value est_Q indicating an estimation of Q's result list length. Before submitting Q_{cand} as a Web query, $est_{Q_{\text{cand}}}$ is internally derived as follows. Let w' be the last added keyword. Hence, the algorithm already knows $est_{Q'}$ for $Q' = Q_{\text{cand}} \setminus \{w'\}$. It 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 is done iff $est_{Q_{\text{cand}}} < 5 \cdot l_{\max}$. The factor 5 in the last inequality was determined experimentally. The experiments revealed that $l_Q \geq est_Q$ for most queries Q and a factor of 5 showed to catch most of the cases where Q is valid but $l_{\max} < est_Q$. Hence, if $est_{Q_{\text{cand}}} < 5 \cdot l_{\max}$ the engine's $l_{Q_{\text{cand}}}$ "adjusts" the internal estimation and is stored as $est_{Q_{\text{cand}}}$.

If $est_{Q_{\text{cand}}} \geq 5 \cdot l_{\max}$, the algorithms do not submit a Web query but store the internally derived $est_{Q_{\text{cand}}}$. In these cases the internal estimation is large enough to ensure that the query Q_{cand} really overflows.

V. Experimental Analysis

Our experiments use search sessions with two or more queries from the AOL 2006 query log. The sessions are obtained by two different detection methods. The first is a temporal method with a 10 minute cut-off (two consecutive queries belong to a session iff they are submitted within 10 minutes) and the second is the geometric method [8] that uses a mix of cut-off time and keyword overlap between consecutive queries. From the obtained sessions stopwords are removed and for every $i \in \{4, 5, \ldots, 10, 15\}$ we randomly choose 1000 sessions containing exactly i keywords for each detection method. Thus, we have 2000 sessions with 4 keywords, 2000 sessions with 5 keywords, etc. We set the bounds $l_{\max} = 100$ and $l_{\min} = 10$. For each session we consider runs of all approaches for Maximum Valid Query and the Query Cover variants. The Microsoft Bing API is used as the Web search engine $\mathcal{S}$.

Table II
Experimental results for Query Cover.

		Number of keywords		
		5	10	15
1	Processed sessions	2 000	2 000	2 000
2	No query cover possible	1 921	1 646	1 072
3	Cover found	79	354	928
4	Avg. *cost* informed Min	11.95	133.14	175.74
5	Avg. *cost* baseline Min	22.81	215.56	268.16
6	**Cost ratio Min**	**0.52**	**0.62**	**0.66**
7	Avg. *cost* informed Max	13.62	135.51	252.22
8	Avg. *cost* baseline Max	23.72	222.21	402.83
9	**Cost ratio Max**	**0.57**	**0.61**	**0.63**
10	Avg. t_{local} informed Min (ms)	2.23	24.15	57.21
11	Avg. t_{Web} informed Min (s)	4.63	53.01	65.43
12	Avg. t_{local} baseline Min (ms)	3.68	29.45	40.16
13	Avg. t_{Web} baseline Min (s)	8.84	85.82	99.84
14	Avg. t_{local} informed Max (ms)	2.41	23.29	62.16
15	Avg. t_{Web} informed Max (s)	5.28	53.95	93.91
16	Avg. t_{local} baseline Max (ms)	2.97	29.11	61.83
17	Avg. t_{Web} baseline Max (s)	9.20	88.47	149.98
18	Avg. Web query time (ms)	387.76	398.13	372.32
19	Avg. $\lvert\mathcal{Q}\rvert$ informed Min	2.00	2.46	2.59
20	Avg. $\lvert\mathcal{Q}\rvert$ baseline Min	2.06	2.56	2.66
21	Avg. query size informed Min	2.58	5.52	8.83
22	Avg. query size baseline Min	2.64	5.56	8.78
23	Avg. $\lvert\mathcal{Q}\rvert$ informed Max	1.82	2.29	2.45
24	Avg. $\lvert\mathcal{Q}\rvert$ baseline Max	1.86	2.35	2.45
25	Avg. query size informed Max	2.66	6.29	10.28
26	Avg. query size baseline Max	2.80	6.53	10.57

A. Query Cover

The results of our experiments for the Query Cover variants can be found in Table II (due to space limits only for sessions with 5, 10, and 15 keywords). The first row states the number of processed sessions. Especially for sessions with few keywords, even the complete query containing all words very often overflows or one of the keywords could not be combined with the others to form a valid query (cf. second row). We filter out such sessions and derive the statistics (rows 4 to 26) just for the remaining sessions (number given in third row). For all these remaining sessions all approaches always find a query cover. In rows 4 and 5 we state the average number *cost* of Web queries needed to solve Minimal Query Cover with and without co-occurrence information. The average ratio of submitted queries is given in row 6 (row 4 divided by row 5). Analogously, rows 7 and 8 contain the average number of queries submitted to solve Maximal Query Cover; the ratio is given in row 9. A visualization of the ratios' behavior is given in Figure 3 (here the ratios for 4–10 and 15 keywords are included). The possible savings compared to the baselines are substantial for both Query Cover variants: 35–50% of the queries with a slight decrease for larger sessions. A potential cause for the decrease is the fact that for larger sessions the absolute and relative number of non-overflowing intermediate queries that have to be submitted to the engine to adjust the internal estimations significantly increases.

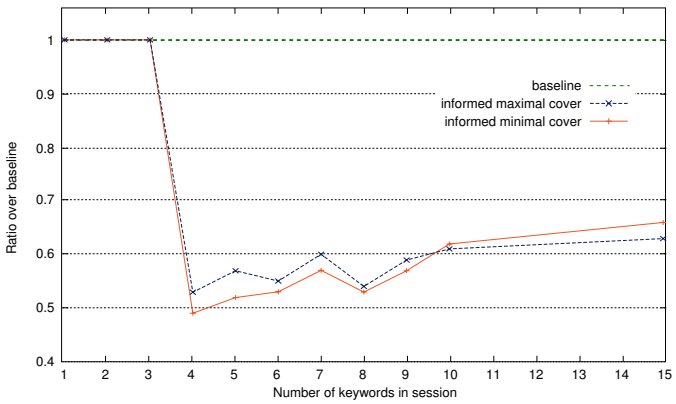

Figure 3. Ratio of queries submitted to solve the QUERY COVER variants via baseline or co-occurrence informed approach.

Rows 10 to 18 comprise the statistics on the time consumption of the algorithms. Note that, as expected, the internal computation time t_{local} (rows 10, 12, 14, and 16) is always orders of magnitude lower than t_{Web} (rows 11, 13, 15, and 17). The co-occurrence informed algorithms are faster than the baselines since they submit significantly fewer queries. The average time of a Web query is given in row 18.

Finally, in rows 19 to 26 we report statistics on the generated query families. The average size of a query and the average number of queries show that the co-occurrence informed algorithms sometimes do not produce the same output as the baselines. This is due to the possible but rare internal overestimations, which hinder the informed approaches to find some of the valid queries that the baselines find. However, the differences are pretty small, and intensive spot checks reveal that the informed algorithms typically produce the same outputs as the baselines.

Altogether, using the co-occurrence graph for internal estimations provides substantial savings in the number of submitted queries; these savings do not harm the quality of $\mathcal{Q}$ in case of the QUERY COVER variants. Compared to the baselines, savings of 35–50% can be expected.

B. Maximum Query

The results of our experiments for MAXIMUM VALID QUERY can be found in Table III (again only for sessions with 5, 10, and 15 keywords). The table's segmentation is similar to that of Table II. The first row states the number of processed sessions. Like in case of the covering variants, sessions with few keywords often do not allow for a maximum query because the complete query containing all words is overflowing (cf. second row). Such sessions are filtered out and the statistics (rows 4 to 13) are derived just for the remaining sessions (number given in third row). For these cases the baseline as well as the co-occurrence informed approach always find a maximum query. In rows 4 and 5 we state the average number *cost* of Web queries needed to solve MAXIMUM VALID QUERY with and without co-occurrence information. The average ratio of submitted queries is given in row 6 (row 4 divided by row 5). A visualization of the

Table III
EXPERIMENTAL RESULTS FOR MAXIMUM QUERY.

		Number of keywords		
		5	10	15
1	Processed sessions	2 000	2 000	2 000
2	No maximum query possible	1 913	1 599	953
3	Maximum query found	87	401	1 047
4	Avg. *cost* informed	10.90	48.15	394.41
5	Avg. *cost* baseline	13.38	70.29	516.46
6	**Cost ratio**	**0.81**	**0.69**	**0.76**
7	Avg. t_{local} informed (ms)	1.20	5.50	76.49
8	Avg. t_{Web} informed (s)	3.91	17.72	132.70
9	Avg. t_{local} baseline (ms)	2.25	7.42	73.34
10	Avg. t_{Web} baseline (s)	4.80	25.86	173.76
11	Avg. Web query time (ms)	359.05	367.94	336.45
12	Avg. size maximum query informed	3.09	7.47	11.93
13	Avg. size maximum query baseline	3.19	7.71	12.34

ratio's behavior is given in Figure 4 (here the ratios for 4–10 and 15 keywords are included). Interestingly, finding a maximum valid query for sessions with 15 keywords causes the submission of more queries than finding a cover with maximal queries. Nevertheless, the possible savings of 15–35% are substantial compared to the baseline. Again, as for the QUERY COVER variants, the slight decrease of possible savings for larger keyword sets is due to the increasing number of non-overflowing queries examined during the search.

The rows 7 to 11 comprise the runtime statistics of the algorithms. Again, the internal computation time t_{local} is orders of magnitude lower than t_{Web} for both approaches. Since the informed approach needs more than 2 minutes for sessions with 15 keywords, the algorithm in this case is applicable only when the user does not need the maximum query immediately after submitting her last query. But it is reasonable to assume that the user also skims through some results of her last query and finding a maximum query can use these "idle" times.

Finally, in rows 12 and 13 we report statistics on the average size of the generated queries. It can be observed that the co-occurrence informed approach does not always produce the same output as the baseline. As for the QUERY COVER variants, this is due to the rare overestimations using the

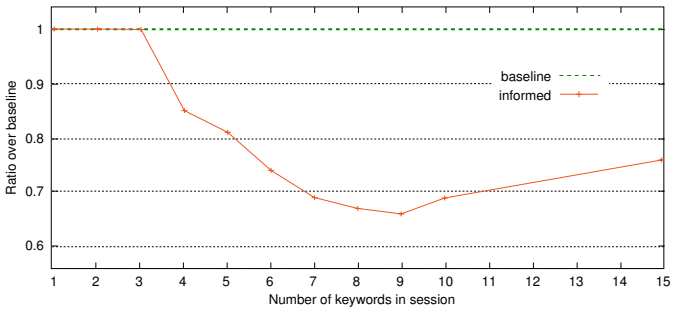

Figure 4. Ratio of queries submitted to solve MAXIMUM VALID QUERY via baseline or co-occurrence informed approach.

internal estimations but the differences are rather small; also here, intensive spot checks reveal that the informed algorithm usually produces the same outputs as the baseline.

We can conclude that using the co-occurrence graph for internal estimations provides substantial savings of 15–35% in the number of submitted queries; these savings do not harm the quality of the found maximum queries.

VI. Conclusion and Outlook

We showed the need for a user-oriented query cost analysis in the process of finding maximum or covering queries against a Web search engine. In such situations the user "plays" against the engine in order to satisfy her information need by submitting Web queries. Our formalization forms the ground for both to define the according problems and to develop search strategies to solve user-cost-oriented optimization versions of the problems. That the co-occurrence informed approaches should be used instead of the uninformed baseline methods is experimentally underpinned by the substantial savings in the number of submitted queries.

Note that the approaches can also be used at search engine site to find maximum or covering query suggestions for search sessions. Such suggestions could have a potential of improving user experience in unsuccessful sessions.

Our investigations leave room for future work. Most importantly, the performance of maximum and covering queries with respect to relevance of the results has to be evaluated. Another interesting task is to use query expansion techniques in order to extend the initial keyword set and thus having an improved diversification of the desired valid queries: especially in case of sessions with few keywords this may lead to a non-overflowing "complete" query. And, finally, the use of potential "sandboxes" from which the co-occurrence probabilities can be derived at zero cost should be analyzed.

References

[1] Ziv Bar-Yossef and Maxim Gurevich. Random sampling from a search engine's index. *Journal of the ACM*, 55(5), 2008.

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

[3] Danushka Bollegala, Yutaka Matsuo, and Mitsuru Ishizuka. An integrated approach to measuring semantic similarity between words using information available on the web. In *Proceedings of HLT 2007*, pages 340–347.

[4] Danushka Bollegala, Yutaka Matsuo, and Mitsuru Ishizuka. Measuring semantic similarity between words using web search engines. In *Proceedings of WWW 2007*, pages 757–766.

[5] David Carmel, Elad Yom-Tov, Adam Darlow, and Dan Pelleg. What makes a query difficult? In *Proceedings of SIGIR 2006*, pages 390–397.

[6] Kevyn Collins-Thompson and Paul N. Bennett. Estimating query performance using class predictions. In *Proceedings of SIGIR 2009*, pages 672–673.

[7] Stephen Cronen-Townsend, Yun Zhou, and W. Bruce Croft. Predicting query performance. In *Proceedings of SIGIR 2002*, pages 299–306.

[8] Daniel Gayo-Avello. A survey on session detection methods in query logs and a proposal for future evaluation. *Information Sciences*, 179(12):1822–1843, 2009.

[9] Claudia Hauff, Djoerd Hiemstra, and Franciska de Jong. A survey of pre-retrieval query performance predictors. In *Proceedings of CIKM 2008*, pages 1419–1420.

[10] Claudia Hauff, Vanessa Murdock, and Ricardo A. Baeza-Yates. Improved query difficulty prediction for the web. In *Proceedings of CIKM 2008*, pages 439–448.

[11] Claudia Hauff, Leif Azzopardi, and Djoerd Hiemstra. The combination and evaluation of query performance prediction methods. In *Proceedings of ECIR 2009*, pages 301–312.

[12] Ben He and Iadh Ounis. Inferring query performance using pre-retrieval predictors. In *Proceedings of SPIRE 2004*, pages 43–54.

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

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

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

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

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] Yun Zhou and W. Bruce Croft. Query performance prediction in web search environments. In *Proceedings of SIGIR 2007*, pages 543–550.