# Applying the User-over-Ranking Hypothesis to Query Formulation*

Matthias Hagen and Benno Stein

Faculty of Media
Bauhaus-Universität Weimar, Germany
<first name>.<last name>@uni-weimar.de

**Abstract** The User-over-Ranking hypothesis states that the best retrieval performance can be achieved with queries returning about as many results as can be considered at user site [21]. We apply this hypothesis to Lee et al.'s problem of automatically formulating a single promising query from a given set of keywords [16]. Lee et al.'s original approach requires unrestricted access to the retrieval system's index and manual parameter tuning for each keyword set. Their approach is not applicable on larger scale, not to mention web search scenarios. By applying the User-over-Ranking hypothesis we overcome this restriction and present a fully automatic user-site heuristic for web query formulation from given keywords. Substantial performance gains of up to 60% runtime improvement over previous approaches for similar problems underpin the value of our approach.

## 1 Introduction

Experienced web search users come up with a whole set of keywords that describe their information need. But if the entire set is submitted as a single query, it is likely that only very few or even no results are returned. On the other hand, single-word queries can usually not satisfy intricate information needs, as such queries are not sufficiently specific. In practice, users then often strive for a longer and more specific query from their keywords, which finally leads to the desired result.

Lee et al. examined the corresponding problem of automatically formulating a single good query from given keywords [16]. Their approach selects the $k$ best keywords according to a learnt ranking function and achieves good performance on TREC topics when given (1) the known relevant documents, (2) full access to an index of the document collection, and (3) a hand-tuned $k$ for each topic—making the algorithm semi-automatic only. Their approach is not applicable in a standard web search scenario since search engines protect their proprietary indexes from direct access. Searches are only possible through interfaces and entail costs; at the very least some non-negligible amount of time is consumed, and larger contingents of queries may entail monetary charges.[1] We overcome the issues of Lee et al.'s approach with a fully-automatic external algorithm (i.e., working at user site against public web search interfaces) that

* Extended version of a paper presented at the TIR 2010 workshop [7].

[1] E.g., \$0.40–\$0.80 per 1000 Yahoo! BOSS queries, `http://www.ysearchblog.com/2011/02/08/latest-on-boss/` (accessed April 16, 2011).

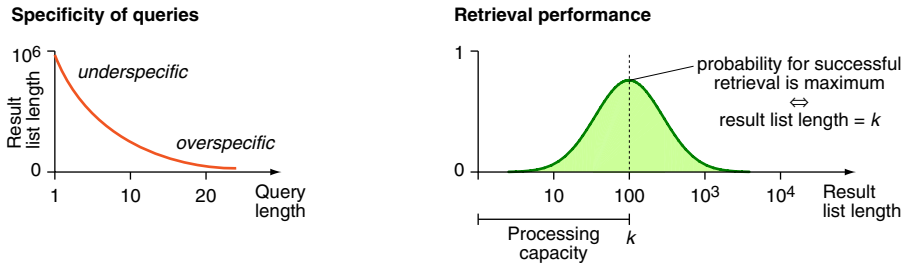


**Figure 1.** Left: a query with few terms and many results is likely to be underspecific; queries with many terms and few results tend to be overspecific. Right: under the User-over-Ranking hypothesis a result list length of the user's capacity $k$ maximizes the retrieval performance [21].

for given keywords finds a web query returning a reasonable number of results. The term *reasonable* deserves closer consideration. Typically, the number of search results a user will consider is constrained by a processing capacity $l_{\max}$, determined by the user's reading time, available processing time, etc. From queries returning more than $l_{\max}$ results only a fraction—typically the top-ranked results—are considered at user site. Often the search engine's ranking works reasonably well to bring up relevant documents to the top even for shorter queries. But what if the first $l_{\max}$ results of a short query don't satisfy the user's information need? The user can never be sure that there are no lower-ranked relevant documents and will try another query containing more or other keywords and thus being more descriptive of her information need. The User-over-Ranking hypothesis states that the user is best served by queries returning in the order of $l_{\max}$ results [21]. In this case, the user can avoid any ranking issues that she cannot influence by simply processing all the documents. The hypothesis is underpinned with a TREC-style experiment showing best performance for queries returning about 100 results [21]. See Figure 1 for an idealized illustration of the hypothesis.

If one follows the User-over-Ranking hypothesis, a central question is: Which subset of the keywords must be chosen to obtain about $l_{\max}$ results? This question is a natural web generalization of Lee et al.'s problem setting. But instead of analyzing the quality of keywords in isolation, the task now is to analyze keyword combinations in order to find queries that are sufficiently specific of a user's information need while not exceeding the user's capacity. The corresponding problem is formalized as follows.

PROMISING QUERY

Given: (1) A set $W$ of keywords.
(2) A query interface for a search engine $\mathcal{S}$.
(3) Upper and lower bounds $l_{\max}$ and $l_{\min}$ on the result list length.

Task: Find a query $Q \subseteq W$ containing the most keywords possible that meets the result list constraints against $\mathcal{S}$.

The convenience lower bound $l_{\min}$ is introduced to rule out queries that return too few result (i.e., setting $l_{\min} = 1$ means that we do not tolerate queries that do not return any result). Note that the PROMISING QUERY formulation is a variant of Lee et al.'s initial setting: given a set $W$, find one good query from $W$. An important difference is that PROMISING QUERY targets the real web setting, where users can only consider a

small amount of results and do not have full access to the engine's index. Obviously, a series of web queries must be submitted to a search engine when solving PROMISING QUERY. As querying consumes time and may entail monetary costs too, we measure the costs at user site with the number of submitted queries and address the optimization problem of minimizing the average number of submitted queries.

### 1.1 Use Cases

Since today's search engines do not suggest promising subsets of a user's keywords, several use-cases can benefit from an external user-site approach. Note that our experimental evaluation in Section 4 is also based on these use cases.

*Known Item Finding.* Assume a user who once had access to a document on the web, forgot to save the document, and now comes up with some keywords that describe the document's content or that occur in the document. Re-finding the desired document has to be done via a web search engine and can be tackled by automatically constructing a good query from the user's set of keywords. Such a query should not return too many results 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 keywords might be "wrong" and should be omitted. Solving an instance of PROMISING QUERY provides a potential way-out.

*Search Session Support.* Assume a web search engine that recognizes sessions of consecutive web queries. Starting with some query, the user submits reformulated queries with varying keywords until she is satisfied or gives up. To support such a user, the engine itself can suggest a promising query returning a reasonable number of results from all the keywords submitted in the session [20]—an instance of PROMISING QUERY. According to the User-over-Ranking hypothesis [21], such promising query suggestions could improve the user's search experience in sessions that did not lead to the satisfaction of the user's information need.

*Empty Result Lists.* A search engine should avoid showing an empty result list on a user's web query. If a query does not produce any hit, an interesting option is to present a largest subset of the keywords that still give a reasonable number of results. According to the User-over-Ranking hypothesis [21], this will raise the probability of satisfying the user's initial information need. Solving an instance of PROMISING QUERY, the engine can provide an appropriate result list instead of an empty page.

### 1.2 Related Work

Besides Lee et al.'s keyword ranking, a lot of research has been done on approaches for better results on better queries. A promising idea is to estimate or to predict a given query's performance [5, 6, 9, 10, 14]. Especially the pre-retrieval predictors, which can be evaluated prior to the actual result retrieval phase, could be interesting for avoiding submission of too many queries. However, the evaluation of most predictors needs access to knowledge the user site does not have in a standard web search scenario. For example, the simplified query clarity predictor [10] needs the total keyword frequencies for the whole corpus—web search engines just return an estimation of the number of documents in the corpus that contain the keyword. The query scope predictor [10] needs

the number of documents in the index—most web search providers stopped publishing it. The mutual information based predictors [14] need the frequency of two keywords in a sliding window with given size over the whole corpus—no engine reports such values. Nevertheless, two approaches on reducing long queries successfully use query quality predictors [14, 17]—but with unrestricted access to the index. The task of long query reduction can be primarily described as the formulation of queries from verbose text, similar to the description parts of TREC topics or queries to medical search engines.

Reducing a large set of keywords to reasonable queries, as is the case in our setting, is also often termed as long query reduction. The interest in the issue of how to handle long queries is on the rise [1, 4, 12, 13, 15], as a significant part of todays typical web queries is becoming longer or "more verbose." In contrast to our setting, most of the existing research approaches assume full access to the system's index; only Huston and Croft use the notion of a black box search engine [11]. However, they focus on a scenario different from ours, namely: finding answers to verbose "wh-"questions in collaborative question answering systems, and they do not analyze the number of submitted web queries. Shapiro and Taksa explicitly deal with the problem setting of formulating queries while considering a bound on the number of results [19]. They suggest a rather simple "open end query formulation." Since their approach does not apply an exhaustive search, it is straightforward to construct situations in which promising queries exist, but the open end approach cannot produce even one of them.

### 1.3 Notation and Basic Definitions

Like in Lee et al.'s setting [16], our starting point for query formulation is a set $W = \{w_1, \ldots, w_n\}$ of keywords (phrases are also allowed). These keywords may be entered by a user or be generated automatically, by an automatic query expansion for example. Subsets $Q \subseteq W$ can be submitted as web queries, with the notion that phrases are included in quotation marks. A web search engine's reply to a query $Q$ consists of a ranked list $L_Q$ of snippets and URLs of documents relevant to the keywords from $Q$, along with an estimation $l_Q$ for the real result list length $|L_Q|$.

Lee et al. try to identify the $k$ "best" keywords in $W$. Their approach relies on the assumption that relevant documents for the information need described by $W$ are known and that $k$ can be manually determined for different $W$. Since this is unrealistic in web search scenarios, we combine the problem setting of automatic query formulation with the User-over-Ranking hypothesis [21]. Hence, our approach will select keywords to form a query that does not return too many results. Not the length of the query is the criterion, which is suggested by Lee et al., but the length of the result list: the PROMISING QUERY problem asks to find a largest subset $Q \subseteq W$ that satisfies $l_{\min} \leq l_Q \leq l_{\max}$ for given constant lower and upper bounds $l_{\min}$ and $l_{\max}$. Requiring $Q$ to be as large as possible ensures $Q$ to be as specific as possible, while the result list constraints reflect the user's capacity, this way accepting the User-over-Ranking hypothesis. Adopting the notation of Bar-Yossef and Gurevich [2], we say that for $l_Q < l_{\min}$ (too few results) the query $Q$ is *underflowing*, whereas for $l_Q > l_{\max}$ (too many results) 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. As for the PROMISING QUERY setting we are interested in the largest maximal queries.

In the process of finding a promising query $Q$, we count the overall number $cost$ of queries that are submitted to the search engine. Since a typical web query takes several hundred milliseconds, the time for internal client site computations will be clearly smaller than the time for submitting the web queries. An approach saving a significant number of web queries will dominate other approaches with respect to runtime, too.

In all query formulation algorithms of this paper, the result list length estimations $l_Q$ of commercial search engines will be used, although they often overestimate the correct numbers. However, the estimations usually respect monotony (queries containing additional keywords have smaller $l$-value, indicating an AND-semantics at search engine site), and the shorter the result list, the better the estimations. Hence, in the range of our user constraints, where we require at most 100 results, they are pretty accurate.

## 2 Baseline Strategies for Promising Queries

The baseline approach can be described as a simple depth-first search on a search tree containing all possible queries; pseudo code listing given as Algorithm 1. A first pre-check removes underflowing keywords (first two lines of the listing) because they cannot be contained in a promising query. Such validity checks for queries always cause a submission to the search engine. A second pre-check (fourth line) ensures that the remaining set $W$ of non-underflowing keywords itself is underflowing, since otherwise $W$ itself is the promising query or no valid query can be found at all. Afterwards, Algorithm 1 invokes an exhaustive search such that it is guaranteed to find a promising query if one exists. Revisiting nodes in the search tree is prohibited by processing the keywords in the order of their indices. 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. Whenever the query underflows, the last keyword is removed and the next one tried. If all keywords have been tried and the resulting query is valid, it is the current candidate to be a promising query. The algorithm then backtracks to other possible paths in the search tree. Pruning is done whenever the current candidate cannot become larger than the currently stored promising query. A valid query that contains more keywords than the current promising query is stored as the new promising query.

**Algorithm 1** Baseline algorithm for PROMISING QUERY

**Input:** keywords $W = \{w_1, \ldots, w_n\}$, result list bounds $l_{\min}$ and $l_{\max}$
**Output:** a largest valid query $Q_{\text{prom}} \subseteq W$

```
for all w ∈ W do
    if {w} is underflowing then W ← W \ {w}
Q_prom ← ∅
if W is underflowing then
    while (W ≠ ∅) ∧ (|W| > |Q_prom|) do
        w ← keyword with lowest index from W
        W ← W \ {w}
        ENLARGE({w}, W)
    output Q_prom
else output {W}
```

```
procedure ENLARGE(query Q, keywords W_left)
    while (W_left ≠ ∅) ∧ (|Q ∪ W_left| > |Q_prom|) do
        w ← keyword with lowest index from W_left
        W_left ← W_left \ {w}
        if Q ∪ {w} is overflowing or valid then
            Q' ← ENLARGE(Q ∪ {w}, W_left)
            if Q' is valid and |Q'| > |Q_prom| then
                Q_prom ← Q'
    return Q
```

Note that Algorithm 1 outputs the lexicographically first promising query with respect to the initial keyword ordering. Here *lexicographically* means the following. Let $Q$ and $Q'$ be two different queries and let $w_{\min}$ be the keyword with lowest index in the symmetric difference $Q \triangle Q' = (Q \cup Q') \setminus (Q \cap Q')$. Then $Q$ comes lexicographically before $Q'$ with respect to the keyword ordering $w_1, \ldots, w_n$ iff $w_{\min} \in Q$. Computing the lexicographically first promising query can be seen as a reasonable approach reflecting the idea that users in their queries first type the keywords that are most descriptive of their information need.

### 2.1 Computing All Promising Queries

Whenever the current candidate can only become as large as the current promising query, Algorithm 1 prunes the search. Instead, a slightly adapted version can check the current candidate enlarged by the full set $W_{\text{left}}$ for validity. If this query is valid, then it forms an additional promising query of the same size as the current promising query and could be stored. If eventually on some other path a valid query is found that is larger than the current promising queries, the stored promising queries are deleted and the set of promising queries is initialized with the then found larger one. Note that this slight change in the baseline yields all promising queries for a given $W$. All of them could be presented to the user, or the lexicographically first one could be selected.

The described technique requires the submission of more queries to the engine. But only in pathological cases, which hardly occur in practice, this will significantly influence the overall performance. Experiments show that computing all promising queries in practice usually requires in the order of $|W|$ additional queries compared to computing just one promising query. These few additional queries can be a worthwhile investment in our heuristics (cf. the corresponding discussion in Section 3).

### 2.2 Co-Occurrence-Informed Baseline

The main drawback of the above uninformed baseline is that it submits all intermediate query candidates to the search engine. To overcome this issue, we improve the approach by informing it with keyword co-occurrence probabilities. A pre-processing step determines $l_{\{w\}}$ for each $w \in W$ and $l_{\{w,w'\}}$ for each pair $w, w' \in W$. Using these values, we store the yield factors $\gamma(w, w') = l_{\{w,w'\}}/l_{\{w\}}$ in a non-symmetric matrix. The yield factor $\gamma(w, w')$ multiplied by $l_{\{w\}}$ gives the yield of web results when the keyword $w'$ is added to the query $\{w\}$. We do not consider the queries needed for obtaining the yield factors. Our rationale is that in case of substantial savings achievable by using the yield factors, a promising future research task is to identify local "sandbox corpora" from which good approximations of web yield factors can be computed at zero cost (e.g., a local index of Wikipedia documents or the ClueWeb collection). Here we show the potential of our yield-factor-informed methods. In order to fairly treat the uninformed baseline, we don't count the baseline's submitted web queries with just one or two keywords in our experimentation.

The informed baseline uses the yield factors to internally estimate the number of returned results for a query candidate. The idea is to check validity without invoking the search engine and to directly enlarge query candidates with overflowing internal estimations.

Let the current query candidate be $Q_{\text{cand}}$ and assume that all queries $Q$ from previous computation steps already have a stored value $est_Q$ indicating an estimation of the length of their result lists. Let $w'$ be the last added keyword. Hence, the informed baseline already knows the value $est_Q$ for $Q = Q_{\text{cand}} \setminus \{w'\}$. It then sets $est_{Q_{\text{cand}}} = est_Q \cdot \text{avg}\{\gamma(w, w') : w \in Q\}$, where avg denotes the mean value. During specific analyses we observed that $l_Q > est_Q$ for most queries $Q$ (i.e., the internal estimations usually significantly underestimate the real number of search results). If this would always be the case, queries with overflowing internal estimations could directly be enlarged. However, our analyses also contained some very rare cases where $Q$ is valid or underflowing but $est_Q > l_{\max}$ (i.e., even the tendency of the internal estimation is wrong). For this reason, the informed heuristic does not blindly follow the internal estimations but only trusts them when $est_{Q_{\text{cand}}} \geq adj \cdot l_{\max}$ for an adjustment factor $adj$. The rationale is that as long as the internal estimations are sufficiently above the validity bound $l_{\max}$, the probability for a wrong validity check based on the internal estimation is negligible. Only when the internal estimation $est_{Q_{\text{cand}}}$ is close to or below the validity bound $l_{\max}$, the current query is submitted to the search engine in order to "adjust" the internal estimation with the search engine's $l_{Q_{\text{cand}}}$. Larger values of $adj$ enlarge the adjustment range and thus guarantee to catch more of the rare cases where $Q$ is valid but $est_Q > l_{\max}$. However, this comes with a larger amount of submitted web queries. Moreover, only huge values of $adj$ can guarantee to return the same promising query as the uninformed baseline. We performed an experimental analysis to compare different reasonable settings of $adj = 1$, 3, 5, and 10. The somehow surprising outcome of these experiments is that a value of $adj = 1$ shows a good overall conformity of the informed baseline's derived promising query with the uninformed baseline's promising query. As setting $adj = 1$ significantly reduces the query cost compared to larger values, the very few differences to the uninformed baseline's results are compensated by a significantly reduced $cost$ resulting in a challenging informed baseline for our heuristics.

## 3 Heuristic Search Strategies

Both the uninformed and the informed algorithms follow the scheme of Algorithm 1 and process the keywords in their initial ordering. We improve upon this ordering and suggest to use co-occurrence information not only to save some of the intermediate queries by internal estimations but also to adopt a heuristic search strategy with a potentially better keyword ordering. Compared to the baselines, the more involved order of processing will save a lot of queries (cf. the experiments in Section 4).

There are two points where a heuristic re-ordering strategy seems to be reasonable: the choices of using the keywords with lowest index as the next to-be-processed keyword (sixth line of Algorithm 1 and second line of procedure ENLARGE). We propose the following two yield-factor-informed re-ordering heuristics.

1. The first heuristic picks as the next keyword $w \in W$ in the sixth line of Algorithm 1 the one with the largest value $l_{\{w\}}$. The rationale is that this will be the keyword with the least commitment: We assume that $w$ together with the next added keywords $W'$ will result in a query $\{w\} \cup W'$ having larger $l$-value than the queries for any other remaining $w' \neq w$.

In the ENLARGE procedure, the heuristic chooses as the best keyword $w \in W_{\text{left}}$ the one having the largest value $\text{avg}\{\gamma(w', w) : w' \in Q_{\text{cand}}\}$. Again, the heuristic's assumption is that this will be the keyword with least commitment (i.e., adding $w$ to the current query candidate $Q_{\text{cand}}$ will decrease the web count the least). Since the heuristic processes the keywords by descending $l$-value and descending yield factor, it is called the *descending heuristic*.

2. The second heuristic is the *ascending heuristic*, which reverses the descending heuristic's ordering. It picks as the next keyword $w \in W$ in the sixth line of Algorithm 1 the one with the smallest value $l_{\{w\}}$. In the ENLARGE procedure, the keyword $w \in W_{\text{left}}$ with the smallest value $\text{avg}\{\gamma(w', w) : w' \in Q_{\text{cand}}\}$ is chosen. The rationale for the ascending heuristic's approach can be best seen in a scenario where some keywords do not "fit" the others: keywords with very small $l$-value or very small $\text{avg}\{\gamma(w', w) : w' \in Q_{\text{cand}}\}$ are often not contained in a promising query. The ascending heuristic's ordering checks these keywords first and thus can weed out them early, while the descending heuristic would unsuccessfully (and costly) try to add them at the end of every search path.

Observe that due to the re-ordering of the keywords the first promising query found by either heuristic may not be the lexicographically first one that the uninformed or the informed baseline compute as their first promising query. This issue can be easily addressed as described in Section 2.1, namely by computing all promising queries and then selecting the lexicographically first among them. An argument for outputting the lexicographically first promising query is that this query probably contains the keywords that are most important for the user as she typed them earlier. Another option is to present all promising queries and let the user select.

## 4 Experimental Analysis

We experimentally compare our two heuristic search strategies to the two baselines. The experimental setting is chosen to reflect the different use cases described in Section 1.1.

### 4.1 Known Item Finding

For the known item finding use case we utilized the corpus from our previous experiments [7]. We crawled a 775 document collection consisting of papers on computer science from major conferences and journals. From each such document—the known items to be found—a number of keywords is extracted by a head noun extractor [3]. We set the bounds $l_{\max} = 100$ (following the findings of the User-over-Ranking hypothesis) and $l_{\min} = 10$. For each document of the test collection we had runs of the baselines and our heuristics with $3, 4, \ldots, 15$ extracted keywords against the Bing API from October 11–23, 2010. A typical web query against the API took about 200–500ms.

Table 1 shows selected results. Especially for sets with few keywords, a promising query is often not possible, since the complete query containing all keywords is still overflowing. The table's statistics are computed for the documents for which a promising query is possible. On these remaining documents all four approaches always find a promising query. Furthermore, the known item—the source document from which the

**Table 1.** Experimental results on the known item use case.

| | Number of keywords | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 5 | 7 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| *Number of documents where* | | | | | | | | | |
| Promising query not possible | 614 | 481 | 338 | 328 | 219 | 155 | 117 | 100 | 86 |
| Promising query found | 161 | 294 | 437 | 447 | 556 | 620 | 658 | 675 | 689 |
| *Average cost (number of submitted queries)* | | | | | | | | | |
| Ascending heuristic | 10.39 | 15.71 | 21.94 | 24.93 | 29.32 | 34.10 | 42.70 | 44.70 | 53.78 |
| Descending heuristic | 9.71 | 15.03 | 22.65 | 25.26 | 35.54 | 45.04 | 73.03 | 91.63 | 130.92 |
| Informed baseline | 10.36 | 16.13 | 24.19 | 27.01 | 36.82 | 47.33 | 71.90 | 70.41 | 108.78 |
| Uninformed baseline | 11.81 | 18.64 | 28.80 | 30.94 | 43.46 | 54.61 | 84.48 | 88.78 | 116.22 |
| *Average cost ratio (basis: uninformed baseline)* | | | | | | | | | |
| Ascending heuristic | 0.88 | 0.84 | **0.76** | **0.81** | **0.67** | **0.62** | **0.51** | **0.50** | **0.46** |
| Descending heuristic | **0.82** | **0.81** | 0.79 | 0.82 | 0.82 | 0.82 | 0.86 | 1.03 | 1.13 |
| Informed baseline | 0.88 | 0.87 | 0.84 | 0.87 | 0.85 | 0.87 | 0.85 | 0.79 | 0.94 |
| *Average size promising query* | | | | | | | | | |
| Ascending heuristic | 3.45 | 5.03 | 6.72 | 7.73 | 8.36 | 8.97 | 9.54 | 9.99 | 10.40 |
| Descending heuristic | 3.49 | 5.03 | 6.77 | 7.76 | 8.39 | 8.97 | 9.50 | 10.07 | 10.41 |
| Informed baseline | 3.50 | 5.02 | 6.79 | 7.83 | 8.38 | 8.98 | 9.53 | 10.07 | 10.55 |
| Uninformed baseline | 3.50 | 5.06 | 6.83 | 7.90 | 8.42 | 9.01 | 9.54 | 10.16 | 10.57 |
| *Average ratio of common result URLs (basis: uninformed baseline)* | | | | | | | | | |
| Ascending heuristic | 0.96 | 0.93 | 0.93 | 0.93 | 0.95 | 0.96 | 0.98 | 0.93 | 0.93 |
| Descending heuristic | 0.98 | 0.96 | 0.96 | 0.96 | 0.95 | 0.96 | 0.96 | 0.97 | 0.93 |
| Informed baseline | 0.99 | 0.98 | 0.98 | 0.98 | 0.99 | 0.98 | 0.98 | 0.97 | 0.98 |

keywords were extracted—always is among the results returned by the search engine for the promising queries. This suggests that promising queries are a reasonable tool to support known item finding.

The average number $cost$ of web queries submitted to solve PROMISING QUERY and the respective ratio of submitted queries compared to the uninformed baseline show that overall the ascending heuristic performs best (smaller $cost$ and smaller ratio indicate better approaches). The ascending heuristic on average submits less than $54$ queries saving more than 50% of the queries compared to the uninformed baseline. Note that the descending heuristic fails to save queries for sets of 14 or more keywords. A possible explanation is that among the extracted keywords a non-negligible part does not fit the rest in the sense that not all extracted keywords describe the same concept.

The quality of the heuristics' promising queries is comparable to the baselines as can be seen by comparing the average size of the generated promising queries and the overlap in the retrieved document URLs. The small differences compared to the uninformed baseline are due to some rare overestimations using the internal estimations, which "hide" some of the queries the uninformed baseline finds (cf. the respective discussion on the adjustment factor setting at the very end of Section 2.2). However, intensive spot checks showed that the heuristics usually produce the same output as the uninformed baseline. This is also supported by the average ratio of common URLs retrieved compared to the uninformed baseline (always above 90%) indicating that the heuristics' results are comparable to the baseline's results.

**Table 2.** Experimental results on the search session use case.

| | Number of keywords | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 5 | 7 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| *Number of sessions where* | | | | | | | | | |
| Promising query not possible | 1939 | 1868 | 1796 | 1719 | 1671 | 1543 | 1387 | 1167 | 903 |
| Promising query found | 61 | 132 | 204 | 281 | 329 | 457 | 613 | 833 | 1097 |
| *Average cost (number of submitted queries)* | | | | | | | | | |
| Ascending heuristic | 11.18 | 19.25 | 33.28 | 44.38 | 63.62 | 73.75 | 81.32 | 97.98 | 102.76 |
| Descending heuristic | 11.41 | 22.11 | 57.03 | 105.32 | 149.83 | 172.08 | 192.39 | 234.70 | 273.18 |
| Informed baseline | 11.90 | 22.35 | 45.07 | 63.40 | 89.74 | 107.11 | 117.02 | 145.83 | 167.92 |
| Uninformed baseline | 14.59 | 29.02 | 73.70 | 110.40 | 156.48 | 175.59 | 198.34 | 227.86 | 250.63 |
| *Average cost ratio (basis: uninformed baseline)* | | | | | | | | | |
| Ascending heuristic | **0.77** | **0.66** | **0.45** | **0.40** | **0.41** | **0.42** | **0.41** | **0.43** | **0.41** |
| Descending heuristic | 0.78 | 0.76 | 0.77 | 0.95 | 0.96 | 0.98 | 0.97 | 1.03 | 1.09 |
| Informed baseline | 0.82 | 0.77 | 0.61 | 0.57 | 0.57 | 0.61 | 0.59 | 0.64 | 0.67 |
| *Average size promising query* | | | | | | | | | |
| Ascending heuristic | 3.31 | 5.10 | 6.57 | 7.48 | 8.48 | 9.56 | 10.79 | 11.83 | 12.78 |
| Descending heuristic | 3.20 | 5.07 | 6.67 | 7.60 | 8.58 | 9.69 | 10.85 | 11.89 | 12.77 |
| Informed baseline | 3.02 | 4.95 | 6.42 | 7.34 | 8.34 | 9.51 | 10.70 | 11.75 | 12.69 |
| Uninformed baseline | 3.34 | 5.22 | 6.77 | 7.70 | 8.67 | 9.73 | 10.87 | 11.92 | 12.83 |
| *Average ratio of common result URLs (basis: uninformed baseline)* | | | | | | | | | |
| Ascending heuristic | 0.97 | 0.93 | 0.92 | 0.91 | 0.92 | 0.94 | 0.95 | 0.97 | 0.97 |
| Descending heuristic | 0.95 | 0.93 | 0.95 | 0.93 | 0.96 | 0.98 | 0.98 | 0.99 | 0.97 |
| Informed baseline | 0.87 | 0.91 | 0.89 | 0.86 | 0.91 | 0.93 | 0.93 | 0.95 | 0.96 |

For the time consumption of the algorithms (not reported in Table 1) we observed the expected behavior: the internal computation time of all approaches is always orders of magnitude lower than the time for web queries (a few milliseconds vs. several seconds or even minutes). Hence, the fastest approach always is the one that submits the fewest queries and the ratio of runtime savings is equivalent to the query savings.

### 4.2 Search Session Support

For the search session use case we utilized a corpus similar to our previous experiments [20]: sessions with at least two queries extracted from the AOL query log [18] using two session detection techniques. One is a temporal method with a 10 minute cutoff (two consecutive queries belong to a session iff they are submitted within 10 minutes) and the other is the cascading method [8] (two consecutive queries belong to a session iff the contained keywords overlap and a time constraint is fulfilled, or if the queries are semantically very similar). Stopwords were removed from the derived sessions and for each method a random sample of 1000 sessions containing $i$ keywords for every $i \in \{4, 5, \ldots, 15\}$ was selected. As before, we set the bounds $l_{\max} = 100$ and $l_{\min} = 10$. For each session we had runs of the algorithms against the Bing API from September 27–October 13, 2010. A typical web query took about 300–600ms.

Table 2 contains the experimental results; the table's organization follows that of Table 1. Again, sessions with few keywords often do not allow for a promising query because the complete query containing all words is still overflowing. Such sessions are

filtered out and the statistics are derived just for the remaining sessions. Note that on these remaining sessions all approaches always find a promising query.

The average number $cost$ of submitted web queries and the respective ratio over the uninformed baseline again show that overall the ascending heuristic performs best. The possible savings seem to converge to about 60% of the queries compared to the uninformed baseline. The descending heuristic does not really improve upon the baselines.

Similar to the known item experiments the quality of the heuristics' promising queries is comparable to the baselines as can be seen by comparing the average query size and the overlap in the retrieved document URLs. As for the time consumption of the algorithms we again observe the expected behavior: the internal computation time of all approaches is always orders of magnitude lower than the web query time. Hence, the fastest approach always is the one that submitted the fewest queries.

### 4.3 Empty Result Lists

For the use case of queries with empty result lists we sampled longer queries from the AOL query log [18]. The log contains 1 015 865 distinct queries with at least 5 and at most 30 keywords. From these, we sampled 497 queries without typos that returned less than 10 results as of October 27–30, 2010 using the Bing API. Empty result lists are modeled by a threshold of 10 returned results (instead of 0) as there exist several mirror pages of the complete AOL query log on the web. Thus, each AOL query with an empty result list back in 2006 today will return several such "mirror" results.

The average query length of the sample is 20.93 keywords (including stopwords). We removed stopwords obtaining an average query length of 12.47 keywords. As before, we set the bounds $l_{\max} = 100$ and $l_{\min} = 10$. Hence, a promising query is possible for all 497 queries (all return less than 10 results). For each query we had runs of the four algorithms against the Bing API from November 04–November 06, 2010. A typical web query in this experiment took about 250–550ms.

All four approaches always find a promising query. On average, the ascending heuristic submitted 92.37 queries, the descending heuristic submitted 207.37, the informed baseline 129.66, and the uninformed baseline 215.41. Hence, the ascending heuristic again obtains the best ratio over the uninformed baseline (about 0.43) and, as before, the ascending heuristic is the fastest among the four approaches. Also the ratio of common URLs (above 0.90 for all approaches) and the size of the promising queries again show that the heuristics' results are comparable to the baselines.

## 5 Conclusion and Outlook

We applied the User-over-Ranking hypothesis to Lee et al.'s query formulation problem and showed the effects of a user-oriented query cost analysis when formulating 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 fully automatic and external algorithms that can be applied against the standard web search engine interfaces.

Altogether, the ascending heuristic should be preferred over the other methods, which is underpinned by experiments for three use cases. The ascending heuristic always comes with substantial savings in the number of submitted queries, whereas these

savings do not impair the quality of the found promising queries. Compared to the uninformed baseline, only 40% of the queries have to be submitted for larger problem instances which is equivalent to a 60% reduced runtime.

A very interesting task for future work is to analyze the use of dedicated sandbox corpora from which yield factors resembling the ones obtained through web queries can be derived at zero cost.

## Bibliography

[1] N. Balasubramanian, G. Kumaran, and V. R. Carvalho. Exploring reductions for long web queries. *Proceedings of SIGIR 2010*, pages 571–578.
[2] Z. Bar-Yossef and M. Gurevich. Random sampling from a search engine's index. *Journal of the ACM*, 55(5), 2008.
[3] K. Barker and N. Cornacchia. Using noun phrase heads to extract document keyphrases. *Proceedings of AI 2000*, pages 40–52.
[4] M. Bendersky and W. B. Croft. Discovering key concepts in verbose queries. *Proceedings of SIGIR 2008*, pages 491–498.
[5] D. Carmel, E. Yom-Tov, A. Darlow, and D. Pelleg. What makes a query difficult? *Proceedings of SIGIR 2006*, pages 390–397.
[6] S. Cronen-Townsend, Y. Zhou, and W. B. Croft. Predicting query performance. *Proceedings of SIGIR 2002*, pages 299–306.
[7] M. Hagen and B. Stein. Search strategies for keyword-based queries. *Proceedings of DEXA 2010 workshop TIR 2010*, pages 37–41.
[8] M. Hagen, T. Rüb, and B. Stein. Query session detection as a cascade. *Proceedings of ECIR 2011 workshop SIR 2011*.
[9] C. Hauff, D. Hiemstra, and F. de Jong. A survey of pre-retrieval query performance predictors. *Proceedings of CIKM 2008*, pages 1419–1420.
[10] B. He and I. Ounis. Inferring query performance using pre-retrieval predictors. *Proceedings of SPIRE 2004*, pages 43–54.
[11] S. Huston and W. B. Croft. Evaluating verbose query processing techniques. *Proceedings of SIGIR 2010*, pages 291–298.
[12] G. Kumaran and J. Allan. Adapting information retrieval systems to user queries. *Information Processing and Management*, 44(6):1838–1862, 2008.
[13] G. Kumaran and J. Allan. Effective and efficient user interaction for long queries. *Proceedings of SIGIR 2008*, pages 11–18.
[14] G. Kumaran and V. R. Carvalho. Reducing long queries using query quality predictors. *Proceedings of SIGIR 2009*, pages 564–571.
[15] M. Lease, J. Allan, and W. B. Croft. Regression rank: Learning to meet the opportunity of descriptive queries. *Proceedings of ECIR 2009*, pages 90–101.
[16] C.-J. Lee, R.-C. Chen, S.-H. Kao, and P.-J. Cheng. A term dependency-based approach for query terms ranking. *Proceedings of CIKM 2009*, pages 1267–1276.
[17] G. Luo, C. Tang, H. Yang, and X. Wei. MedSearch: a specialized search engine for medical information retrieval. *Proceedings of CIKM 2008*, pages 143–152.
[18] G. Pass, A. Chowdhury, and C. Torgeson. A picture of search. In *Proceedings of Infoscale 2006*, paper: 1.
[19] J. Shapiro and I. Taksa. Constructing web search queries from the user's information need expressed in a natural language. *Proceedings of SAC 2003*, pages 1157–1162.
[20] B. Stein and M. Hagen. Making the most of a web search session. *Proceedings of WI-IAT 2010*, pages 90–97.
[21] B. Stein and M. Hagen. Introducing the user-over-ranking hypothesis. *Proceedings of ECIR 2011*, pages 503–509.