# Webis at the TREC 2010 Sessions Track

Matthias Hagen , Benno Stein, and Michael Völske

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

**Abstract** In this paper we provide an overview of the Webis group's two-phase approach to the TREC 2010 Sessions track. In a preprocessing phase the queries are segmented to highlight contained concepts. In the final retrieval phase we treat Carnegie Mellon's ClueWeb search engine as a black box and apply the MAXIMUM QUERY framework.

## 1 Introduction

The TREC 2010 Sessions track offered the opportunity to apply our developed techniques for user experience improvement during web search sessions. Our framework is inspired by the observation that 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—or keyphrases—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.

The track itself has two tasks: (1) to improve retrieval performance for a given query by using the user's previous queries from the same session, and (2) to improve retrieval performance over an entire query session instead of a single query. Our framework for these tasks consists of two parts. In a first step we use a query segmentation approach from [HPSB10] to automatically detect and highlight concepts and phrases within the queries. In the second step—the retrieval process itself—we adopt a user perspective against an existing ClueWeb09 search engine and apply the MAXIMUM QUERY framework from [SH10].

We apply the user perspective for the following reason. Experience shows that in many cases a user's first web query for her information need is answered reasonably well by existing commercial search engines (i.e., the query brings up an appropriate result). In case that the first query was not successful, commercial search engines provide different means of support (e.g., query expansion for queries returning lots of hits or spelling correction for queries returning no hits due to typos). However, there is no real session support yet such that respective techniques could also be implemented at user site (e.g., in a proxy process that handles a user's web queries).

Our approach has a more combinatorial flavor than current search engine's user support techniques but it is easily combinable with existing technology. We suggest

to derive 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). 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 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 $k$, 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. Consistent with our User-over-Ranking hypothesis [SH11], 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. That queries returning about $k$ many results indeed can improve retrieval performance is underpinned by our experimental justification for the User-over-Ranking hypothesis [SH11]

Hence, from the user's perspective, a maximum query contains a possible description of the information need and offers the chance to check all the results. However, finding a maximum query "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 apply an algorithm. To be applicable at user site the algorithm is of *external* nature (i.e., it only uses the search engine's interfaces). The search engine is handled as a black box, acting like an oracle that answers queries. There is no need for the user to know the underlying retrieval model or implementation details. The black box we use for the Sessions track is Carnegie Mellon's ClueWeb search engine[1].

The paper is organized as follows. In Section 2 we describe the applied query segmentation process. The actual retrieval process is presented in Section 3. Achieved results of our system are shown in Section 4. A discussion and some concluding remarks follow in Section 5.

## 2 Preprocessing phase

The TREC 2010 Sessions track queries contained no phrase information. Hence, we decided to preprocess each single query by automatically detecting segments of words that should have been highlighted as phrases for improved retrieval. As a query segmentation procedure we use the $s^s$-weighting scheme [HPSB10].

The basic and major assumption is that phrases contained in web queries really exist on the web. The straightforward idea then is to use the web itself as a corpus to detect

[1] http://boston.lti.cs.cmu.edu:8085/clueweb09/search/

potential query segments. The approach uses phrase occurrence frequencies to decide which phrases are more likely segments than others. The largest obtainable collection of web phrases is the Google $n$-gram corpus [BF06]; it contains $n$-grams of length 1 to 5 from the 2006 Google index along with their frequencies.

A query $q$ is viewed as a sequence $(w_1, w_2, \ldots, w_n)$ of $n$ keywords. A valid segmentation $S$ for $q$ is a sequence of disjunct segments $s$, each a subsection of $q$, whose concatenation equals $q$. There are $2^{n-1}$ possible valid segmentations for $q$, and $(n^2 - n)/2$ possible segments that contain at least two keywords from $q$. We derive a score for each of the possible segmentations as follows. First, the $n$-gram frequency $count(s)$ for each possible segment $s$ is retrieved. The frequencies of $n$-grams up to $n = 5$ can be obtained directly from the corpus; we use an index of the Google $n$-gram corpus in an inverted file from [SPT10][2]. For longer $n$-grams up to $n = 9$, estimations are made analog to the set-based method described in [TP08]. Having these counts at hand, all valid segmentations are enumerated systematically, and for each segmentation $S$ a score is computed according to the following function:

$$score(S) = \sum_{s \in S, |s| \geq 2} |s|^{|s|} \cdot count(s).$$

The factor $|s|^{|s|}$ gives significant weight to longer segments compared to short ones in order to compensate the power law distribution of occurrence frequencies on the web. For example, "`new york`" has a much larger count than "`new york times`". The exponential scoring function should help to avoid segmentations like "`new york`" "`times`". For a query $q$ we choose from all possible valid segmentations the segmentation $S$ that maximizes $score(S)$. This simple approach is competitive with more involved methods, as evaluation shows [HPSB10].

## 3 Retrieval phase

In the real retrieval phase we treat the Carnegie Mellon ClueWeb search engine as a black box and submit segmented queries. When processing a query we adopt the MAXIMUM QUERY framework [SH10] that works as follows.

### 3.1 Basic setting

Starting point is a set $W = \{w_1, \ldots, w_n\}$ of keywords and keyphrases (obtained by the preprocessing), where the indices correspond to the order of the keywords in the original query. Subsets $Q \subseteq W$ can be submitted as queries (complete phrases would be highlighted) assuming the AND notion that requires all keywords from $Q$ to be contained in every document. The ClueWeb engine's reply to a query $Q$ consists of the head of an exhaustive, ranked list $L_Q$ of documents containing all the keywords from $Q$, and an estimation $l_Q$ for the result list length $|L_Q|$.

[2] A corresponding service that can also handle wildcard queries etc. is freely available online: http://www.netspeak.cc/

The task of MAXIMUM QUERY is 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 $l_{\max}$. As for the Sessions track we set $l_{\min} = 1$ (we do not tolerate empty result lists) and $l_{\max} = 1\,000$ (1 000 results should be reported per run). The size constraint on $Q$ ensures $Q$ to be as specific of the user's information need as possible while the result list constraints reflect the desired capacity (in the TREC Sessions track case no more than 1 000 results would be considered per run). Adopting the notation from [BG08] 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 *maximal* iff adding any keyword from $W \setminus Q$ results in an underflowing query. The largest maximal queries are the *maximum* queries—the "target" of MAXIMUM QUERY.

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 1.

**Table 1.** Keyword document relationship in the example scenario.

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

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.

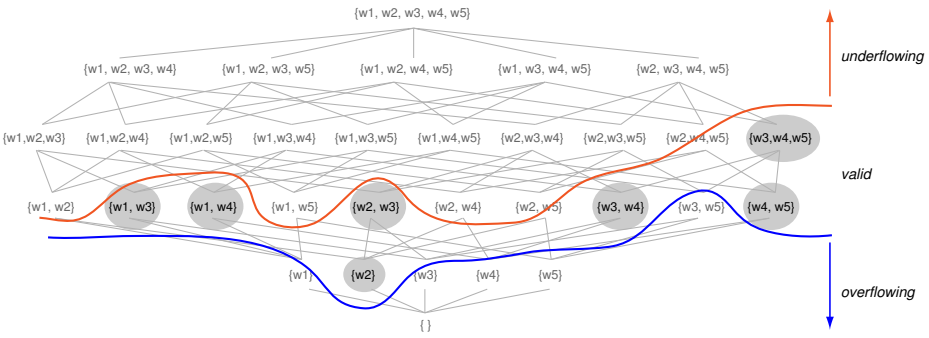


**Figure 1.** Hypercube of possible queries in the example scenario.

An example of an overflowing query is $\{w_3, w_5\}$ (six results), whereas $\{w_1, w_5\}$ is underflowing (two results). We have the four maximal valid queries $\{w_1, w_3\}$, $\{w_1, w_4\}$, $\{w_2, w_3\}$, and $\{w_3, w_4, w_5\}$ corresponding to the upper border in Figure 1. In our example scenario the unique maximum element is $\{w_3, w_4, w_5\}$.

### 3.2 Algorithm for finding maximum queries

The pseudo-code listing of our approach to find maximum queries is given as Algorithm 1. A first pre-check 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 ClueWeb search engine. A second pre-check (line 4) ensures that the remaining set $W$ of non-underflowing keywords itself is underflowing, since otherwise $W$ itself is maximum or no valid query can be found at all. For the rather short queries and sessions of the TREC 2010 Sessions track very often no keyword was removed as underflowing and the complete query mostly still overflows. On such sessions we just report the first 1 000 results for the complete query $W$. As for the few other sessions we applied the following main part of our algorithm.

The basic idea can be characterized as a depth-first search on a search tree containing all possible queries. Revisiting nodes in the tree is prohibited by processing the keywords in the order of their indices. The algorithm starts trying to find a maximal

**Algorithm 1** Algorithm for MAXIMUM QUERY

**Input:** $W = \{w_1, \ldots, w_n\}$, $l_{\min}$, 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** $Q$

valid query containing the first keyword $w_1$. It 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 maximum 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 maximum query. A valid query that contains more keywords 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.

Note that Algorithm 1 outputs the lexicographically first maximum query. 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 $Q \triangle Q' = (Q \cup Q') \setminus (Q \cap Q')$. We say that $Q$ comes lexicographically before $Q'$ with respect to the keyword ordering $w_1, \ldots, w_n$ iff $w_{\min} \in Q$. Computing the lexicographically first maximum query is a reasonable approach as it reflects the idea that users in their queries first type the keywords that are most descriptive of their information need. Hence, for the Sessions track we always use the lexicographically first maximum query.

However, it would not be difficult to compute all maximum queries submitting a few additional web queries and then select one of the maximum queries.

## 4 Evaluation

The evaluation for both tasks of the Sessions track is done by comparing three rankings. One ranking has to be given for each of the two queries from the originally provided 150 two-query sessions. A third ranking could incorporate the knowledge that both queries form a session.

### 4.1 Our runs

We have submitted two runs of our system, both with the query segmentation preprocessing that highlights contained phrases. As for the individual single queries from each session, both runs process the first and the second queries analogous as follows (respective result lists referred to as $query_1$ and $query_2$). For each single query we computed a maximum query. In case of more than one maximum query the lexicographically first with respect to the initial keyword ordering of the given queries is chosen. However, as already described earlier, the provided 300 queries are rather short such that very rarely terms were excluded from the queries in order to get back 1 000 results (chosen as the upper bound as the track required submission of 1 000 results). From the maximum queries for the first queries 138 out of 150 contain all phrases; for the second queries 141 out of 150 contain all phrases. For all queries the top 1 000 results where used for the run when available.

The two runs of our system treated the individual single queries in the same way but used a different scheme for the queries corresponding to the complete sessions. We had an unweighted and a weighted run (respective ranked lists referred to as $session_u$ for unweighted and $session_w$ for weighted) that worked as follows. In $session_u$ we

applied the described MAXIMUM QUERY framework for the complete set $W$ of phrases from both queries. In case of more than one maximum query the lexicographically first with respect to the initial keyword ordering of the given queries is chosen. Again, very often all keywords together still overflow; 128 out of 150 maximum queries contain all phrases. For all queries the top 1 000 results where used for the result list when available.

In $session_w$ we additionally pay attention to sessions with generalizations (second query originated by deleting keywords from the first) or specializations (second query contains additional keywords compared to first). Therefore, we weighted the query phrases as follows (and derived maximum queries with respect to that weighting). For phrases just appearing in the first query a term weight in Carnegie Mellon's ClueWeb09 search engine's Indri query language is set to 0.5; in case that it is present just in the second query, the weight is set to 2.0; for terms in both queries the weight is 1.0. The idea is that by submitting a second query the user figured out that her first query did not satisfy her information need and thus the new keywords in the second query should be treated as more important. Note that the weighting scheme also influences the "drifting" sessions that are neither specializations nor generalizations. Again, very often all keywords together still overflow; 128 out of 150 maximum queries contain all phrases. For all queries the top 1 000 results where used for then result list when available.

### 4.2 Evaluation metrics

The runs were evaluated on 136 query sessions for which NIST provided relevance judgments. No judgments were given for the topics 24, 30, 35, 36, 58, 100, 114, 118, 120, 126, 130, and 136. Based on the provided relevance judgments, three evaluation metrics are used: nsDCG@10, nsDCG_dupes@10, and nDCG@10. The nsDCG@10 metric [JPDN08] is computed as:

$$\sum_{r=1}^{10} \frac{2^{rel(r,query_1)} - 1}{(\log_2(r+1)) * (\log_4(1+3))} + \sum_{r=1}^{10} \frac{2^{rel(r,query_2/session)} - 1}{(\log_2(r+10+1)) * (\log_4(2+3))},$$

where $rel(r, query_1)$ is the relevance of the document in rank $r$ in the $query_1$ result list and $rel(r, query_2/session)$ is the relevance of the document at rank $r$ in the $query_2$ or the $session$ result lists. The nsDCG_dupes@10 metric is similar to nsDCG@10 except that duplicate documents in the top-10 ranks of the $query_2$ or the $session$ result lists that appeared in the top-10 ranks of the $query_1$ result list are considered non-relevant. Note that nsDCG@10 and nsDCG_dupes@10 evaluate the entire session and thus for each one of the two metrics there is an evaluation score for $query_1 \rightarrow query_2$ and an evaluation score for $query_1 \rightarrow session$ (cf. Tables 2 and 3).

The nDCG@10 is the nDCG metric implemented as

$$\sum_{r=1}^{10} \frac{2^{rel(r,query_1/query_2/session)} - 1}{(\log_2(r+1))}$$

for the three ranked lists for $query_1$, $query_2$, or $session$ in isolation.

### 4.3 Our obtained results

**Table 2.** Results for **nsDCG@10** averaged over all 136 topics.

| | $query_1 \rightarrow query_2$ | $query_1 \rightarrow session$ |
|---|---|---|
| unweighted | 0.1796 | 0.1674 |
| weighted | 0.1796 | 0.1724 |
| median all systems | 0.2044 | 0.1784 |
| max all systems | 0.2488 | 0.2375 |

Our nsDCG@10 results and for comparison the median and max of all systems are given in Table 2. It is interesting to note that our runs as well as the median and maximum of all systems have better results $query_1 \rightarrow query_2$ than for $query_1 \rightarrow session$. This seems to suggest that the techniques to incorporate knowledge on the whole session do not really work for the test cases. This is not that surprising at least for our systems. Our approaches were not developed having such rather short queries and sessions in mind; the targeted use case are longer sessions with more keywords (cf. the discussion in Section 5).

The nsDCG@10 for $query_1 \rightarrow session$ is the official metric for Task (2)—retrieval performance over an entire query session. Our overall averaged nsDCG@10 $query_1 \rightarrow session$ score on all 136 topics is better for our weighted run compared to the unweighted run. Nevertheless, it is still slightly below the median of all systems. However, analyzed topicwise, our weighted run beats the median of all systems on 84 of 136 topics (61.76%) and we achieve the best (maximum) performance of all systems on 4 topcis (2.94%). This is quite surprising as we did not expect to perform that well on the rather short queries and sessions that do not really fit the use case of our system.

**Table 3.** Results for **nsDCG@10_dupes** averaged over all 136 topics.

| | $query_1 \rightarrow query_2$ | $query_1 \rightarrow session$ |
|---|---|---|
| unweighted | 0.1859 | 0.1654 |
| weighted | 0.1859 | 0.1748 |
| median all systems | 0.2067 | 0.1869 |
| max all systems | 0.2449 | 0.229 |

The behavior for the duplicate free version of nsDCG (results in Table 3) is similar to the pure nsDCG. Our weighted variant improves over the unweighted one but again is slightly below the median of all systems. However, seen topicwise, our weighted $session_w$ run beats the median of all systems on 86 out of 136 topics (63.24%) and yields the best (maximum) performance of all systems for 5 topics (3.68%).

As for Task (1)—retrieval performance for a given query by using the user's previous query—, performance can be evaluated by comparing the scores of $query_1 \rightarrow query_2$ and $query_1 \rightarrow session$ for the nsDCG metrics. Our results on the $session$ result lists are worse compared to the $query_2$ lists for both nsDCG metrics but this is

the same with the median or the maximum of all systems. Again it should be noted that we expect our framework to perform better for longer sessions and queries where computing maximum queries makes more sense (cf. the discussion in Section 5).

As to compare the results "query-wise" we give our obtained nDCG values in Table 4.

**Table 4.** Results for **nDCG@10** averaged over all 136 topics.

| | $query_1$ | $query_2$ | $session$ |
|---|---|---|---|
| unweighted | 0.1638 | 0.2014 | 0.1621 |
| weighted | 0.1638 | 0.2014 | 0.1776 |
| median all systems | 0.1894 | 0.2144 | 0.17 |
| max all systems | 0.2354 | 0.2658 | 0.2602 |

Again, our system (weighted and unweighted) and the median and best over all systems perform best on the second query alone not incorporating knowledge about the first query. At least for our system we hypothesize that we could perform better on longer sessions with more keywords; which gives rise to the following discussion and concluding remarks.

## 5 Discussion

As can be seen from the evaluation, our approach performs comparable to the median of all systems. This is quite encouraging, however, at least on average the retrieval performance tables show that yet our approach yields no improvement by considering two queries as a session compared to just processing the second query alone. But note that this holds for the median and maximum of all systems as well.

One reason might be the rather short queries and the session length of two queries. At least our approach is designed for longer sessions with more queries and keywords. In case of the 150 sessions of the 2010 TREC Sessions track often the complete query containing all keywords returned enough hits such that computing the maximum query does not remove any keywords. Furthermore, some topics are quite similar to each other (e.g., the various queries for the `obama family tree`) or rather "artificial" (`to be or not to be` meant to check stopword techniques?!).

Nevertheless, evaluating session processing techniques and not just single query retrieval is an important task from our perspective. As the Sessions track addresses exactly this problem it opens the evaluation perspective on a very interesting research area and we hope that it will be continued. An interesting future task might however involve longer and more diverse sessions.

## References

[BF06] Thorsten Brants and Alex Franz. Web 1T 5-gram Version 1. Linguistic Data Consortium LDC2006T13, Philadelphia, 2006.

[BG08] Ziv Bar-Yossef and Maxim Gurevich. Random sampling from a search engine's index. *Journal of the ACM*, 55(5), 2008.

[HPSB10] Matthias Hagen, Martin Potthast, Benno Stein, and Christof Bräutigam. The Power of Naïve Query Segmentation. In Fabio Crestani, Stéphane Marchand-Maillet, Hsin-Hsi Chen, Efthimis N. Efthimiadis, and Jacques Savoy, editors, *Proceeding of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2010, Geneva, Switzerland, July 19-23, 2010*, pages 797–798. ACM, July 2010.

[JPDN08] Kalervo Järvelin, Susan L. Price, Lois M. L. Delcambre, and Marianne Lykke Nielsen. Discounted cumulated gain based evaluation of multiple-query IR sessions. In *Advances in Information Retrieval , 30th European Conference on IR Research, ECIR 2008, Glasgow, UK, March 30-April 3, 2008. Proceedings*, pages 4–15, 2008.

[SH10] Benno Stein and Matthias Hagen. Making the Most of a Web Search Session. In *2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT 2010)*, pages 90–97, August 2010.

[SH11] Benno Stein and Matthias Hagen. Introducing the User-over-Ranking Hypothesis. In *Advances in Information Retrieval, Proceedings of the 33rd European Conference on Information Retrieval (ECIR 2011)*, Lecture Notes in Computer Science, Berlin Heidelberg New York, 2011. Springer. (to appear).

[SPT10] Benno Stein, Martin Potthast, and Martin Trenkmann. Retrieving Customary Web Language to Assist Writers. In Cathal Gurrin, Yulan He, Gabriella Kazai, Udo Kruschwitz, Suzanne Little, Thomas Roelleke, Stefan M. Rüger, and Keith van Rijsbergen, editors, *Advances in Information Retrieval, Proceedings of the 32nd European Conference on Information Retrieval (ECIR 2010)*, volume 5993 of *Lecture Notes in Computer Science*, pages 631–635, Berlin Heidelberg New York, 2010. Springer.

[TP08] Bin Tan and Fuchun Peng. Unsupervised query segmentation using generative language models and Wikipedia. In *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008*, pages 347–356, 2008.