Efficient Two-Sided Error-Tolerant Search

Hannah Bast Marjan Celikik
University of Freiburg, Germany
{bast,celikik}@informatik.uni-freiburg.de

ABSTRACT

We consider fast two-sided error-tolerant search that is robust against errors both on the query side (type *alogrithm*, find documents with *algorithm*) as well as on the document side (type *algorithm*, find documents with *alogrithm*). We show how to realize this feature with an index blow-up of 10% to 20% and an increase in query time by a factor of at most 2. We have integrated our two-sided error-tolerant search into a fully-functional search engine, and we provide experiments on three data sets of different kinds and sizes.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Search process; H.3.4 [**Systems and Software**]: Performance evaluation (efficiency and effectiveness)

General Terms

Algorithms, Experimentation, Measurement, Performance

Keywords

Error-Tolerant Search, Fuzzy Search

1. INTRODUCTION

Users make mistakes when typing their queries into a search engine, either because they mistype the keywords or because they do not know the correct spelling of the keywords they have in mind. It is therefore one of the most important features of a search engine to be robust against such mistakes on the side of the user.

But the text collections indexed by the search engine also contain misspellings, either made by those who have written the documents or incurred in the process of converting the documents in electronic form (e.g. OCR). For example, even in the extremely carefully hand-maintained DBLP metadata, there are 17 papers with *probalistic* in the title. For an application like literature search, it is an equally important feature as the one above to find these 17 papers when typing the correct word *probabilistic* as (part of) a query.

In this short paper, we will show how to realize such two-sided error-tolerant search both efficiently and with good error robustness.

2. RELATED WORK

KEYS'10, June 6, 2010, Indianapolis, Indiana, USA.

There is an enormous body of literature on finding words similar to a given query word in a large lexicon efficiently; see [2] for a list of references. The straightforward way to use this for text search that is robust against spelling errors is to replace each query word by a *disjunction* (OR) of its similar words. We call this algorithm BASELINE here. BASELINE is all but efficient: on a collection like Wikipedia, almost any word has hundreds of reasonably similar words; see Section 4. A simple remedy would be to consider only selected similar words (for example, the most frequent ones), but that obviously affects recall.

There is surprisingly little work on efficient high-recall error-tolerant search, as described in the introduction. And there is hardly any work that deals with both kinds of errors: on the side of the query and on the side of the documents. A prior work from [2] deals only with errors of the second type. In [3], both kinds of errors are addressed by suggesting an alternative to BASELINE. However, there is an improvement only for queries with multiple keywords and then only when one of the keywords is much more specific than the others. For many typical queries, their approach therefore is not much faster than BASELINE.

3. OUR ALGORITHM

Like in [2], we measure the similarity of two words w and v by their *normalized edit distance*, denoted by $\mathrm{ED}_0(w, v)$. We define $\mathrm{ED}_0(w, v) := ED(w, v)/\max\{|w|, |v|\}$, where ED is the standard edit distance. Note that $\mathrm{ED}_0(w, v)$ is always in $[0, 1]$, reaching 0 iff $w = v$, and reaching 1 iff w and v have no character in common. A convenient property of $\mathrm{ED}_0(w, v)$ is that the edit distance threshold is automatically increased on longer and decreased on shorter words.[1]

3.1 Clustering of the Vocabulary

The *vocabulary* of a document collection is the set of distinct words occurring in it. Our algorithm is based on a *clustering* of the words in the vocabulary. The clusters may *overlap*, that is, a word may belong to more than one cluster. Before we can state what makes a good clustering, we need some definitions.

DEFINITION 1. *Consider a fixed clustering and a fixed similarity threshold δ. For a query word q, that may or may not be in the vocabulary, let $S_q = \{w : \mathrm{ED}_0(q, w) \leq \delta$ be the set of words similar to q. An exact cover of S_q is a set of clusters whose union C contains S_q. An approximate cover does not necessarily contain all of S_q. The* recall *of a cover (exact or approximate) is defined as $|S_q \cap C|/|S_q|$. The* precision *of a cover (exact or approximate) is defined as $|S_q \cap C|/|C|$. Recall and precision of a cover are 100% if and only if $S_q = C$. Finally, we define the* cover index *of a cover as the number of clusters in the cover.*

[1]The normalized edit distance threshold for our experiments was set to 0.28, as suggested in [2]

Now for each query word q we want to find a cover of S_q with the following properties: (C1) cover index as small as possible, (C2) recall as large as possible, and (C3) precision as large possible. As will see below, the cover index determines the blow-up in query time compared to an ordinary (non error-tolerant) query. Finally, we also want the *frequency-weighted cluster overlap* to be as small as possible (C4). This overlap is defined as $\sum_w \mathrm{tf}_w \cdot c_w / \sum_w \mathrm{tf}_w$, where the sum is over all words w in the vocabulary, tf_w is the total number of occurrences of a word w, and c_w is the number of clusters containing w. As we will see this ratio is exactly the blow-up in index space compared to an index for ordinary (not error-tolerant) search.

3.2 Using the Clustering

Assume we have a clustering with favorable properties (C1) - (C4) from above. Our algorithm consists of the following components:

Indexing Time. For each occurrence of a word w, say `house`, determine the ids of the clusters containing `house`, say 165 and 9823, and then add the corresponding words `C:165:house` and `C:9823:house` to the index (this is done once for every occurrence of the word).

Query Time. For each query word `q`, compute the cluster ids of a cover of S_q, say 4325 and 23, and then replace that query word by the disjunction of prefix queries `C:4325:*|C:23:*`, where `*` and `|` are the prefix and disjunction operators, respectively. To process the transformed queries, which are conjunctions of disjunctions of prefix queries, we use the HYB index from [1].

Minimal Cover Index. The following greedy strategy finds a cover of S_q with minimal cover index: consider all clusters containing words from S_q; start by picking the cluster containing the largest number of words from S_q, preferring smaller clusters in the case of ties; take this cluster and the covered words out of consideration and iterate.

3.3 Computing a Clustering

Our clustering algorithm is based on the following two observations. First, valid words (which are usually frequent) are the natural cluster centroids of their spelling variants. Second, to minimize the frequency weighted cluster overlap, c_w should depend on the frequency of w, with very frequent words assigned to a single cluster.

Given the sorted list of words in order of decreasing term frequency we pick the first word as cluster centroid and compute and mark each word in its neighborhood of similar words. We proceed by picking the next unmarked word in the list and stop once a cut-off frequency is reached. Next, we assign the closest c_w centroids to each marked word w, where c_w depends on tf_w. Finally, very infrequent words get a special treatment, in that their clusters are forced to be non-overlapping. With these ideas we achieve a clustering with average cover index of 4 or less (C1), an average recall always over 90% (C2), an average precision of over 60% (C3), and index space blow-up of less than 20% (C4) (as well as various trade-offs among the four). Note that precision of the cover is the least critical of all four aspects, since we rank our results by the edit distance of the matching words to the respective query words.

4. EXPERIMENTS

Our experiments were carried out on three datasets of different sizes and kinds. Our largest collection is the January 2009 dump of the English *Wikipedia*, with 9.3 million articles and a vocabulary of 8.5 million words. *DBLP-Full* is a collection of 31,211 computer science articles, with a vocabulary of 1.0 million words and many OCR errors. *DBLP-Meta* is a relatively clean collection of 1.3 million BibTeX entries. We generated 1000 (frequent and infrequent) single-keyword queries by randomly selecting keywords from the collection, and applying between 0 and 3 random edits.

4.1 Query Processing Overhead

Table 1 shows that the blow-up in query time of our two-sided error-tolerant search compared to ordinary (not error-tolerant) search is about a factor of 2 or less, quite independently of the size and kind of the collection, where for BASELINE this factor is more than 20. To verify that the results are not affected by our implementation, we repeated the experiment on BASELINE with the Lucene[2] search engine and obtained similar running times.

Table 1: Average query times.

	DBLP-Meta	DBLP-Full	Wikipedia
ORDINARY	1.26 ms	6.8 ms	61.0 ms
OUR METHOD	2.0 ms	11.2 ms	112.6 ms
BASELINE	11.9 ms	121.6 ms	1468.2 ms
avg. clusters	2.4	2.2	4.1
avg. similar words	20	70	77

4.2 Quality

Table 2 shows that we can achieve an average cover recall of over 95% and an average cover precision of about 60% for the query processing overhead from Table 1. Results for DBLP-Metadata are not shown but are similar. We note that we could achieve 100% cover recall at the cost of doubling the query time. As already noted in Section 3, a medium cover precision is fine for us since the result hits are ranked by the edit distance of the matching words to the respective query words.

Table 2: Cover precision and recall

length	4-6		7-9		10+		overall	
frequency	high	low	high	low	high	low	high	low
DBLP-Full								
recall	.91	.97	.98	.99	.98	.99	.96	.99
precision	.64	.60	.71	.64	.73	.60	.69	.61
Wikipedia								
recall	.90	.90	.95	.96	.97	.99	.93	.95
precision	.60	.50	.61	.54	.68	.68	.63	.57

4.3 Index Size and Construction Time

Table 3 shows that the blow-up in index space is at most 20% for all three collections. Indexing time roughly doubles, since our clustering algorithm takes about as long as the rest of the indexing process. We work to improve on this in the future.

Table 3: Index sizes.

	DBLP-Meta	DBLP-Full	Wikipedia
ORDINARY	91 MB	414 MB	8.4 GB
OUR METHOD	115 MB	472 MB	9.5 GB

5. REFERENCES

[1] H. Bast and I. Weber. Type less, find more: fast autocompletion search with a succinct index. In *SIGIR*, pages 364–371, 2006.

[2] M. Celikik and H. Bast. Fast error-tolerant search on very large texts. In *SAC*, pages 1724–1731, 2009.

[3] S. Ji, G. Li, C. Li, and J. Feng. Efficient interactive fuzzy keyword search. In *WWW*, pages 371–380, 2009.

[2] `http://lucene.apache.org/`