Efficient Interactive Query Expansion with CompleteSearch

Holger Bast
MPI für Informatik
Saarbrücken, Germany
bast@mpi-inf.mpg.de

Debapriyo Majumdar
IBM India Research Lab
Bangalore, India
debapriyo@in.ibm.com

Ingmar Weber
MPI für Informatik
Saarbrücken, Germany
iweber@mpi-inf.mpg.de

ABSTRACT

We present an efficient realization of the following interactive search engine feature: as the user is typing the query, words that are related to the last query word and that would lead to good hits are suggested, as well as selected such hits. The realization has three parts: (i) building clusters of related terms, (ii) adding this information as artificial words to the index such that (iii) the described feature reduces to an instance of prefix search and completion. An efficient solution for the latter is provided by the CompleteSearch engine, with which we have integrated the proposed feature. For building the clusters of related terms we propose a variant of latent semantic indexing that, unlike standard approaches, is completely transparent to the user. By experiments on two large test-collections, we demonstrate that the feature is provided at only a slight increase in query processing time and index size.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Query formulation, Search process; H.3.1 [**Content Analysis and Indexing**]: Indexing Methods, Thesauruses; H.5.2 [**User Interfaces**]: Interaction styles

General Terms

Design, Experimentation, Human Factors, Performance

Keywords

Index Building, Interactive, Query Expansion, Synsets, Wikipedia, WordNet

1. INTRODUCTION

Adding related terms to the query in order to increase recall is an old and much-researched technique commonly known as *query expansion*. A recent, very good overview is provided in [3]. Our work is different from the traditional approach in two respects.

Efficiency The standard way to implement query expansion is to replace each query word by the *disjunction (OR)* of its related terms, and then *merge* the individual inverted lists [3]. Since each step of such a merge is logarithmic in the number of lists, expansion is typically limited to a few important words. Sophisticated top-k techniques, like in [10], try to prune expansion words which are unlikely to lead to an improved ranking.

Interactivity & context-sensitivity Our feature interactively suggests words related to the word currently being typed, and the suggestions are ranked by their ability to lead to good hits together with the preceding part of the query. For example, for the query `russia metal` from Figure 1, `aluminum` is high up in the list of words related to `metal`, because there are many news articles about aluminum production in Russia. Similar interactive features have been discussed in the literature [7], but with the focus on effectiveness and not on efficiency.

We do not claim novelty for any of these individual points (efficient, interactive, context-sensitive). However, we have not seen them presented in combination, and we want to stress the simplicity with which we realize this feature here, provided that an efficient prefix completion mechanism is available. We remark that synonym search has already been mentioned in [2], but the idea from Section 3, which is key for an efficient implementation, was missing at that time.

2. PREFIX COMPLETION SEARCH

The following prefix search and completion operation was presented in [2]: given a prefix p (the partially typed last query word) and a set of documents D (the hits for the preceding part of the query), compute a ranked list of words starting with p and occurring in D, as well as a ranked list of documents from D containing words starting with p. The contribution of [2] was a new index data structure, coined HYB, for efficient support of this operation. The main idea behind HYB is to store unions of inverted lists of groups of words that share a common prefix in a highly compressed format.

3. QUERY EXPANSION VIA PREFIX COMPLETION

The key idea is to add the information we have about related terms as artificial words to the index such that the

CIKM'07, November 6–8, 2007, Lisboa, Portugal.

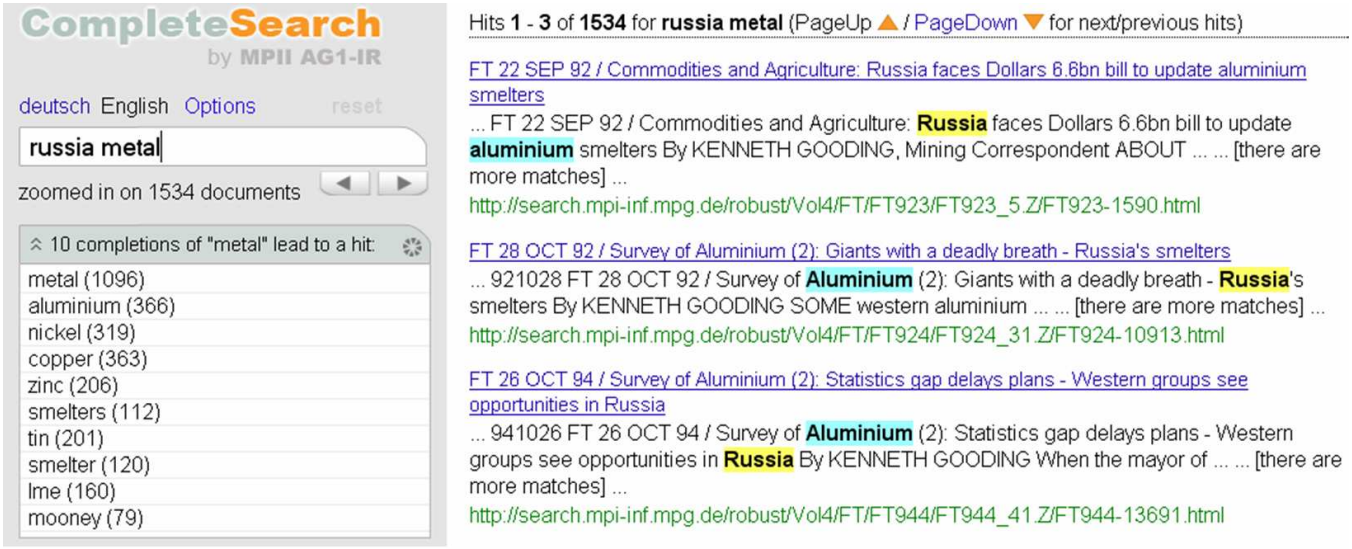

Figure 1: The proposed feature, integrated into the CompleteSearch engine. The box under the search field shows words related to the last query word, metal in this case, that would lead to good hits, and to the right the best such hits are displayed. Both related words and hits are updated automatically after each keystroke.

feature we want becomes a matter of a few prefix completion operations. We realize this as follows:

0. As input, assume we have *clusters* of related terms. These clusters may overlap, for example the word `case` may be in one cluster together with `cover`, `shell`, etc. as well as in another cluster together with `box`, `chest`, etc.
1. For each occurrence of a term `<t>` that occurs in a cluster with id `<id>`, add the artificial term `s:<id>:<t>`, in the same document and at exactly the same position.

   (Document position is important for proximity and phrase search. We remark that the HYB index does not mind several terms at the same position in the same document.)

   Also add, but only *once* for each such term and in a special document that is used for no other purpose, the artificial term `s:<t>:<id>`. The number of words in this document is just the total size of all clusters.
2. For a given query `<q1> ... <ql> <p>`, we then realize our feature via one or two prefix completion operations as follows: We first check whether `<q1> ... <ql> s:<p>` has a unique completion of the form `s:<t>:<id>`. If so, this completion gives us the id `<id>` of the cluster containing `<t>`. Then the query `<q1> ... <ql> s:<id>:` gives us the desired completions and hits.

   (Note that the part `<q1> ... <ql>` is evaluated only once, just after its last letter being typed, and stored by a cache-like mechanism from then on; see [2] for details.)

4. TERM CLUSTERS

We implemented and tested our feature for two collections: the TREC Robust collection (1.5 GB, 556,078 documents), and the English Wikipedia (8 GB, 2,863,234 documents). For each of the collections we used a different method to derive the clusters of related terms, one unsupervised and one supervised. The following two subsections give details on each of the methods. The results of the experiments are given in Section 5.

4.1 Unsupervised approach - spectral method

For the Robust collection, we used a completely unsupervised approach based on the technique from [1], as follows.

1. Since Eigenvector computations on large matrices are very expensive, we first removed common stopwords, and restricted ourselves to a set of frequent nouns, which we identified using the TnT tagger[1] [4]. We then ran Porter's stemming[2][8] algorithm on these nouns and got a set of 10,098 terms to work with.
2. We obtained a set of related term pairs from these 10,098 terms using the smoothness test described in [1]. We used a high smoothness threshold to select term pairs to ensure that no two unrelated (or not closely related) terms qualify as related terms. Figure 2 shows the kind of term-term relations extracted this way.
3. We used the Markov Clustering Algorithm (MCL) algorithm[3] from [11] to derive clusters from the list of term pairs, as required by our approach.

Note that this approach makes the result of this unsupervised learning algorithm, and its effect on the search results, completely transparent to the user. In contrast, methods in the spirit of latent semantic indexing [5] are often criticized for their incomprehensibility on the side of the user concerning why a certain document show up high in the ranking. It would be interesting to verify the significance of this difference in a user study.

[1] `http://www.coli.uni-saarland.de/~thorsten/tnt`
[2] http://www.tartarus.org/~martin/PorterStemmer/perl.txt
[3] `http://micans.org/mcl`

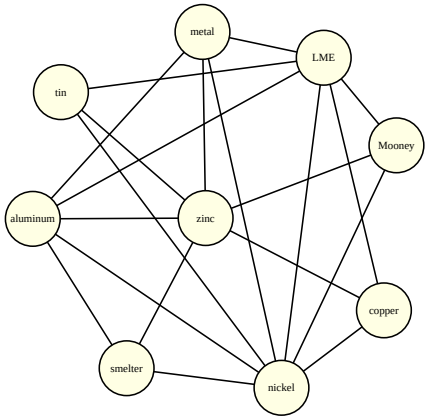

Figure 2: One of the clusters of related terms automatically obtained from the Robust collection. Edges present in the graph denote term-term relations found by the smoothness test from [1]. The cluster itself was then found using the clustering algorithm from [11]. Indeed, all the terms in the cluster are closely related: most of them are different metals, LME stands for London Metal Exchange, and Richard Mooney is the author of several articles regarding the general topic of metal.

4.2 Supervised approach

For the Wikipedia, we made a straightforward use of WordNet [6] to obtain clusters of related terms. Namely, we put two words that occur somewhere in Wikipedia in the same cluster if and only if they share the same most frequent synset. E.g., for the term "car" the synset corresponding to "auto","automobile","machine" and "motorcar" was used, but *not* the ones corresponding to "railcar" or "gondola". Table 1 shows all synsets for the term "car". This heuristic leads to only about 30% more tokens in the index. Using *all* synets, would spoil both efficiency and usefulness of our feature.

Furthermore, we only used single terms and ignored compound nouns in open form ("lawn tennis"), as we build our index for individual terms.[4] The descriptions and the example phrases for the synsets were also not used, as they do not explicitly contain any synonymy information.

5. EXPERIMENTS

We integrated the described feature with the CompleteSearch engine, and measured its efficiency on two query sets. The first query set is derived from the 200 "old"[5] queries (topics 301-450 and 601-650) of the TREC Robust Track in 2004 [12]. For the second query set, we started with 100 random queries, generated as follows: For each query, we picked a random document and sampled 1 to 5 terms (with a mean of 2.2 and a median of 2, which are realistic values for web search queries [9]) according to their tf-idf values.

For both query sets, these raw queries were then "typed" from left to right, using a minimal prefix length of 3. So the raw query "cult lifestyles" would yield the autocompletion queries `cul`, `cult`, `cult lif`, `cult life` and so on. Additionally, whenever for a prefix `<p>` the query `s:<p>` led to a *unique* term cluster with id `<id>`, we added an OR (for which we use the "|") with the prefix `s:<id>:`. E.g., one autocompletion query in the sequence for "airport security" is `airport|s:399: secu|s:385:`.

All experiments were run on a machine with two 2.8 GHz AMD Opteron processors (two cores each, but only one of them used per run), with 16 GB of main memory, operating in 32-bit mode, running Linux.

[4]Inclusion of such compound nouns is theoretically possible, but it was not implemented for this study.

[5]They had been used in previous years for TREC.

1. car, auto, automobile, machine, motorcar (a motor vehicle with four wheels; usually propelled by an internal combustion engine) "he needs a car to get to work"
2. car, railcar, railway car, railroad car (a wheeled vehicle adapted to the rails of railroad) "three cars had jumped the rails"
3. car, gondola (the compartment that is suspended from an airship and that carries personnel and the cargo and the power plant)
4. car, elevator car (where passengers ride up and down) "the car was on the top floor"
5. cable car, car (a conveyance for passengers or freight on a cable railway) "they took a cable car to the top of the mountain"

Table 1: A complete list of the WordNet synsets for the noun "car". For our experiments, we assigned each word only to its most frequent synset, so for "car" we used the first set in the list above. The term "machine" would in the end not be used as a synonym for car, as its most frequent synset refers to a different concept.

Query set	Average	90%-tile	99%-tile	Max
Robust (all)	32 ms	90 ms	375 ms	970 ms
- normal	22 ms	55 ms	329 ms	970 ms
- synonyms	57 ms	129 ms	385 ms	655 ms
Wikipedia (all)	64 ms	238 ms	614 ms	1218 ms
- normal	42 ms	128 ms	569 ms	1218 ms
- synonyms	35 ms	356 ms	799 ms	841 ms

Table 2: Breakdown of processing times for both of our query sets. For "normal" queries there was no synonymy information to be used.

Table 2 shows that, by using the term clusters, the average processing time increases by roughly 50% (but not more) with respect to queries without synonymy information, and it is still well within the limits of interactivity. Somewhat surprisingly, the maximum processing time is *lower* for the queries *with* synonymy information. This is because the queries which take the longest to process are those with a very unspecific last query word, for example, `cont`. Such words tend to have more than one completion for which synonymy information is available, and in that case our interface, as described above, does not show any related terms, but only syntactic completions.

6. REFERENCES

[1] Bast, H., and Majumdar, D. Why spectral retrieval works. In *SIGIR* (2005), pp. 11–18.

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

[3] Billerbeck, B. *Efficient Query Expansion*. PhD thesis, RMIT University, 2005.

[4] Brants, T. Tnt – a statistical part-of-speech tagger. In *ANLP* (2000), pp. 224–231.

[5] Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W., and Harshman, R. A. Indexing by latent semantic analysis. *JASIS 41*, 6 (1990), 391–407.

[6] Fellbaum, C., Ed. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[7] Fonseca, B. M., Golgher, P. B., Pôssas, B., Ribeiro-Neto, B. A., and Ziviani, N. Concept-based interactive query expansion. In *CIKM* (2005), pp. 696–703.

[8] Porter, M. F. An algorithm for suffix stripping. *Program 14*, 3 (1980), 130–137.

[9] Spink, A., Jansen, B. J., Wolfram, D., and Saracevic, T. From e-sex to e-commerce: Web search changes. *IEEE Computer 35*, 3 (2002), 107–109.

[10] Theobald, M., Schenkel, R., and Weikum, G. Efficient and self-tuning incremental query expansion for top-k query processing. In *SIGIR* (2005), pp. 242–249.

[11] van Dongen, S. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, 2000. http://micans.org/mcl.

[12] Voorhees, E. Overview of the Trec 2004 Robust retrieval track. In *TREC* (2004).