

Lucene and Juru at Trec 2007: 1-Million Queries Track

Einat Amitay, David Carmel, Doron Cohen
IBM Haifa Research Lab
Haifa 31905, Israel

Email: {einat,carmel,doronc}@il.ibm.com

ABSTRACT

Lucene is an increasingly popular open source search library. Its out-of-the-box search quality for TREC data however falls significantly behind other systems.

By modifying Lucene's scoring function, we were able to improve its search quality for TREC data, and it is now comparable to Juru, our home-brewed search engine.

This work reports our experiments and the modifications involved - namely better normalization of document length and term frequency measures.

1. INTRODUCTION

Our TREC experiments this year were focused on the One-Million Queries track, examining the scoring function of Lucene, an Apache open-source search engine [8]. We used our home-brewed search engine, Juru [6], to compare with Lucene on the 10K track's queries over the gov2 collection.

Lucene¹ is an open-source library for full-text indexing and searching in Java. The default scoring function of Lucene implements the cosine similarity function, while search terms are weighted by the tf-idf weighting mechanism. Equation 1 describes the default Lucene score for a document d with respect to a query q :

$$Score(d, q) = \sum_{t \in q} tf(t, d) \cdot idf(t) \cdot boost(t, d) \cdot norm(d) \quad (1)$$

where

- $tf(t, d)$ is the term frequency factor for the term t in the document d .
- $idf(t)$ is the inverse document frequency of the term.

¹We used Lucene Java [1], the first and main Lucene project. Lucene defines a file format, and hence there are ports of Lucene to other languages. Throughout this work, "Lucene" refers to "Apache Lucene Java".

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

- $boost(t, field, d)$ is the boost of the field of t in d , as set during indexing.
- $norm(d)$ is the normalization value, given the number of terms within the document.

For more details on Lucene scoring see [2].

Our main goal in this work was to experiment with the scoring mechanism of Lucene in order to bring it to the same level as the state-of-the-art ranking formulas such as OKAPI [9] and the SMART scoring model [5]. In order to study the scoring model of Lucene in full details we ran Juru and Lucene over the 150 topics of the tera-byte tracks, and over the 10K queries of the one-million queries track of this year. We then modified Lucene's scoring function to include better document length normalization, and a better term-weight setting according to the SMART model. Equation 2 describes the term frequency (tf) and the normalization scheme used by Juru, based on SMART scoring mechanism, that we followed to modify Lucene scoring model.

$$tf(t, d) = \frac{\log(1+freq(t, d))}{\log(1+avg(freq(d)))} \quad (2)$$
$$norm(d) = \frac{1}{\sqrt{0.8avg(\#uniqueTerms) + 0.2\#uniqueTerms(d)}}$$

For both Juru and Lucene we used the same HTML parser to extract content of the Web documents of the gov2 collection. In addition, for both systems the same anchor text extraction process took place. Anchors data was used for two purposes: 1) Set a document static score according to the number of in-links pointing to it. 2) Enrich the document content with the anchor text associated with its in-links. Last, for both Lucene and Juru we used a similar query parsing mechanism, which included stop-word removal, synonym expansion and phrase expansion.

In the following we describe these processes in full details and the partial results we already have.

2. ANCHOR TEXT EXTRACTION

Extracting anchor text is a necessary task for indexing Web collections: adding text surrounding a link to the indexed document representing the linked page. With inverted indexes it is often inefficient to update a document once it was indexed. In Lucene, for example, updating an indexed document involves removing that document from the index and then (re) adding the updated document. Therefore, anchor extraction is done prior to indexing, as a global computation step. Still, for large collections, this is a nontrivial task. While this is not a new problem, describing a timely extraction method may be useful for other researchers.

2.1 Extraction method

Our GOV2 input is a hierarchical directory structure of about 27,000 compressed files, each containing multiple documents (or pages), altogether about 25,000,000 documents. Our output is a similar directory structure, with one compressed anchors text file for each original pages text file. Within the file, anchors texts are ordered the same as pages texts, allowing indexing the entire collection in a single combined scan of the two directories. We now describe the extraction steps.

- **(i) Hash by URL:** Input pages are parsed, emitting two types of result lines: page lines and anchor lines. Result lines are hashed into separate files by URL, and so for each page, all anchor lines referencing it are written to the same file as its single page line.
- **(ii) Sort lines by URL:** This groups together every page line with all anchors that reference that page. Note that sort complexity is relative to the files size, and hence can be controlled by the hash function used.
- **(iii) Directory structure:** Using the file path info of page lines, data is saved in new files, creating a directory structure identical to that of the input.
- **(iv) Documents order:** Sort each file by document numbers of page lines. This will allow to index pages with their anchors.

Note that the extraction speed relies on serial IO in the splitting steps (i), (iii), and on sorting files that are not too large in steps (ii), (iv).

2.2 GOV2 Anchors Statistics

Creating the anchors data for GOV2 took about 17 hours on a 2-way Linux. Step (i) above ran in 4 parallel threads and took most of the time: 11 hours. The total size of the anchors data is about 2 GB.

We now list some interesting statistics on the anchors data:

- There are about 22 million anchors for about 25 million pages, an average of about 9 anchors per page.
- Maximum number of anchors of a single page is 1,275,245 - that many GOV2 pages are referencing the page *www.usgs.gov*.
- 17% of the pages have no anchors at all.
- 77% of the pages have 1 to 9 anchors.
- 5% of the pages have 10 to 99 anchors.
- 159 pages have more than 100,000 anchors.

Obviously, for pages with many anchors only part of the anchors data was indexed.

2.3 Static scoring by link information

The links into a page may indicate how authoritative that page is. We employed a simplistic approach of only counting the number of such in-links, and using that counter to boost up candidate documents, so that if two candidate pages agree on all quality measures, the page with more incoming links would be ranked higher.

Static score (SS) values are linearly combined with textual scores. Equation 3 shows the static score computation for a document d linked by $in(d)$ other pages.

$$SS(d) = \sqrt{\min(1, \frac{in(d)}{400})} \quad (3)$$

It is interesting to note that our experiments showed conflicting effects of this static scoring: while SS greatly improves Juru's quality results, SS have no effect with Lucene. To this moment we do not understand the reason for this difference of behavior. Therefore, our results include static scoring by in-links count for Juru but not for Lucene.

3. QUERY PARSING

We used a similar query parsing process for both search engines. The terms extracted from the query include single query words, stemmed by the Porter stemmer which pass the stop-word filtering process, lexical affinities, phrases and synonyms.

3.1 Lexical affinities

Lexical affinities (LAs) represent the correlation between words co-occurring in a document. LAs are identified by looking at pairs of words found in close proximity to each other. It has been described elsewhere [7] how LAs improve precision of search by disambiguating terms.

During query evaluation, the query profile is constructed to include the query's lexical affinities in addition to its individual terms. This is achieved by identifying all pairs of words found close to each other in a window of some predefined small size (the sliding window is only defined within a sentence). For each LA=(t1,t2), Juru creates a pseudo posting list by finding all documents in which these terms appear close to each other. This is done by merging the posting lists of t1 and t2. If such a document is found, it is added to the posting list of the LA with all the relevant occurrence information. After creating the posting list, the new LA is treated by the retrieval algorithm as any other term in the query profile.

In order to simulate lexical affinities by Lucene we used the Lucene's SpanNearQuery class which matches spans of the query terms which are within a given window in the text.

3.2 Phrase expansion

The query is also be expanded to include the query text as a phrase. For example, the query *'oil industry history'* is expanded to *'oil industry history. "oil industry history"'*. The idea is that documents containing the query as a phrase should be biased compared to other documents. The posting list of the query phrase is created by merging the postings of all terms, considering only documents containing the query terms in adjacent offsets and in the right order. Similarly to LA weight which specifies the relative weight between an LA term and simple keyword term, a phrase weight specifies the relative weight of a phrase term.

Phrases are simulated by Lucene using the PhraseQuery class.

3.3 Synonym Expansion

The 10K queries for the one-million track are strongly related to the gov domain hence many abbreviations are found in the query list. For example, the u.s. states are

usually referred by abbreviations (e.g. “ny” for New-York, “ca” or “cal” for California). However, in many cases relevant documents refer to the full proper-name rather than its acronym. Thus, looking only for documents containing the original query terms will fail to retrieve those relevant documents.

We therefore used a synonym table for common abbreviations in the gov domain to expand the query. Given a query term with an existing synonym in the table, expansion is done by adding the original query phrase while replacing the original term with its synonym. For example, the query “ca veterans” is expanded to “ca veterans. California veterans”.

3.4 Lucene query example

To demonstrate our choice of query parsing, for the original topic – *U.S. oil industry history*, the following Lucene query was created:

```
oil industri histori
(
  spanNear([oil, industri], 8, false)
  spanNear([oil, histori], 8, false)
  spanNear([industri, histori], 8, false)
)^4.0
"oil industri histori"~1^0.75
```

The result Lucene query illustrates some aspects of our choice of query parsing:

- “U.S.” is considered a stop word and was removed from the query text.
- Only stemmed forms of words are used.
- Default query operator is OR.
- Words found in a document up to 7 positions apart form a lexical affinity. (8 in this example because of the stopped word.)
- Lexical affinity matches are boosted 4 times more than single word matches.
- Phrase matches are counted slightly less than single word matches.
- Phrases allow fuzziness when words were stopped.

4. LUCENE SCORE MODIFICATION

Our TREC quality measures for the GOV2 collection revealed that the default scoring of Lucene [2] is inferior to that of Juru. (Detailed run results are given in Section 6.)

We were able to improve Lucene’s scoring by changing the scoring in two areas: document length normalization and term frequency (tf) normalization.

Lucene’s default length normalization [3] is given in equation 4, where L is the number of words in the document.

$$lengthNorm(L) = \frac{1}{\sqrt{L}} \quad (4)$$

The rationale behind this formula is to prevent very long documents from “taking over” just because they contain many terms, possibly many times. However a negative side

effect of this normalization is that long documents are “punished” too much, while short documents are preferred too much. The first two modifications below remedy this further.

4.1 Sweet Spot Similarity

Here we used the document length normalization of “Sweet Spot Similarity” [4]. This alternative Similarity function is available as a Lucene’s “contrib” package. Its normalization value is given in equation 5, where L is the number of document words.

$$lengthNorm(L) = \frac{1}{\sqrt{steepness*(|L-min|+|L-max|-(max-min))+1}} \quad (5)$$

We used $steepness = 0.5$, $min = 1000$, and $max = 15,000$. This computes to a constant norm for all lengths in the $[min, max]$ range (the “sweet spot”), and smaller norm values for lengths out of this range. Documents shorter or longer than the sweet spot range are “punished”.

4.2 Pivoted length normalization

Our pivoted document length normalization follows the approach of [10], as depicted in equation 6, where U is the number of *unique* words in the document and $pivot$ is the average of U over all documents.

$$lengthNorm(L) = \frac{1}{\sqrt{(1 - slope) * pivot + slope * U}} \quad (6)$$

For Lucene we used $slope = 0.16$.

4.3 Term Frequency (tf) normalization

The tf-idf formula used in Lucene computes $tf(t \text{ in } d)$ as $\sqrt{freq(t, d)}$ where $freq(t, d)$ is the frequency of t in d . We modified this to take into account the average term frequency in d , similar to Juru, as shown in formula 7.

$$tf(t, d) = \frac{\log(1 + freq(t, d))}{\log(1 + avgFreq(d))} \quad (7)$$

This distinguishes between terms that highly represent a document, to terms that are part of duplicated text. In addition, the logarithmic formula has much smaller variation than Lucene’s original square root based formula, and hence the effect of $freq(t, d)$ on the final score is smaller.

4.4 Query fewer fields

Throughout our experiments we saw that splitting the document text into separate fields, such as *title*, *abstract*, *body*, *anchor*, and then (OR) querying all these fields performs poorer than querying a single, combined field. We have not fully investigated the causes for this, but we suspect that this too is due to poor length normalization, especially in the presence of very short fields.

Our recommendation therefore is to refrain from splitting the document text into fields in those cases that all the fields are actually searched at the same time.

While splitting into fields allows easy boosting of certain fields at search time, a better alternative is to boost the terms themselves, either statically by duplicating them at indexing time, or perhaps even better dynamically at search

time, e.g. by using term payload values, a new Lucene feature. In this work we took the first, more static approach.

5. QUERY LOG ANALYSIS

This year’s task resembled a real search environment mainly because it provided us with a substantial query log. The log itself provides a glimpse into what it is that users of the .gov domain actually want to find. This is important since log analysis has become one of the better ways to improve search quality and performance.

For example, for our index we created a stopword list from the most frequent terms appearing in the queries, crossed with their frequency in the collection. The most obvious characteristic of the flavor of the queries is the term *state* which is ranked as the 6th most frequent query term with 342 mentions, and similarly it is mentioned 10,476,458 times in the collection. This demonstrates the bias of both the collection and the queries to content pertaining to state affairs. Also in the queries are over 270 queries containing *us*, *u.s.*, *usa*, or the phrase “*united states*” and nearly 150 queries containing the term *federal*. The fact is that the underlying assumption in most of the queries in the log is that the requested information describes US government content. This may sound obvious since it is after all the .gov collection, however, it also means that queries that contain the terms *US*, *federal*, and *state* actually restrict the results much more than required in a collection dedicated to exactly that content. So removing those terms from the queries makes much sense. We experimented with the removal of those terms and found that the improvement over the test queries is substantial.

Term	Frequency in query log	Frequency in index
of	1228	Originally stopped
in	838	Originally stopped
and	653	Originally stopped
for	586	Originally stopped
the	512	Originally stopped
state	342	10476458
to	312	Originally stopped
a	265	Originally stopped
county	233	3957936
california	204	1791416
tax	203	1906788
new	195	4153863
on	194	Originally stopped
department	169	6997021
what	168	1327739
how	143	Originally stopped
federal	143	4953306
health	135	4693981
city	114	2999045
is	114	Originally stopped
national	104	7632976
court	103	2096805
york	103	1804031
school	100	2212112
with	99	Originally stopped

Table 1: Most frequent terms in the query log.

The log information allowed us to stop general terms along with collection-specific terms such as *US*, *state*, *United States*, and *federal*.

Another problem that arose from using such a diverse log was that we were able to detect over 400 queries which contained abbreviations marked by a dot, such as *fed.*, *gov.*, *dept.*, *st.* and so on. We have also found numbers such as in street addresses or date of birth that were sometimes provided in numbers and sometimes in words. Many of the queries also contained abbreviations for state names which is the acceptable written form for referring to states in the US.

For this purpose we prepared a list of synonyms to expand the queries to contain all the possible forms of the terms. For example for a query *Orange County CA* our system produced two queries *Orange County CA* and *Orange County California* which were submitted as a single query combined of both strings. Overall we used a list of about 250 such synonyms which consisted of state abbreviations (e.g. *RI*, *Rhode Island*), common federal authorities’ abbreviations (e.g. *FBI*, *FDA*, etc.), and translation of numbers to words (e.g. *1*, *one*, *2*, *two*, etc.).

Although we are aware of the many queries in different languages appearing in the log, mainly in Spanish, we chose to ignore cross-language techniques and submit them as they are. Another aspect we did not cover and that could possible have made a difference is the processing of over 400 queries that appeared in a *wh question* format of some sort. We speculate that analyzing the syntax of those queries could have made a good filtering mechanism that may have improved our results.

6. RESULTS

Search quality results of our runs on the 150 topics of the tera-byte tracks are given in table 2. For a description of the various options see section 4.

Run	MAP	P@5	P@10	P@20
1. Juru	0.313	0.592	0.560	0.529
2. Lucene Default Similarity	0.214	0.409	0.390	0.383
3. Lucene Sweet Spot Similarity	0.273	0.593	0.565	0.527
4. Lucene Pivot Length Norm	0.284	0.572	0.540	0.503
5. Lucene TF avg norm	0.194	0.404	0.373	0.370
6. Lucene Pivot + TF avg norm	0.294	0.587	0.542	0.512
7. Lucene Pivot + Sweet Spot	0.306	0.627	0.589	0.543

Table 2: Search Quality Comparison.

Run 1, the only Juru run, was used as a target. Run 2, the default Lucene configuration, was used as a base line. Comparing the first two lines shows that Lucene’s default behavior does not perform very well, with MAP of 0.214 comparing to Juru’s MAP of 0.313.

Lucene’s sweet spot similarity run performs much better than the default Lucene behavior, bringing MAP to 0.273. Note that the only change here is a different document length normalization.

Replacing the document length normalization with a pivoted length one in run 4 improves MAP further to 0.284.

The best one is definitely run 7, that combines the document length normalization of Lucene’s sweet spot similarity with normalizing the tf by the average term frequency in a

document. This combination brings Lucene's MAP to 0.306, and even outperforms Juru for the P@5, P@10, and P@20 measures.

7. SUMMARY

We showed that Lucene's out of the box search quality for TREC data is significantly inferior to other search engines that participate in TREC, and in particular comparing to our search engine Juru.

We were able to improve Lucene's search quality as measured for TREC data by a better choice of document length normalization and by normalizing tf values by document's average term frequency.

Application developers using Lucene can easily adopt the document length normalization part of our work, simply by a different similarity choice. However for applying the tf normalization some changes in Lucene code would be required.

8. REFERENCES

- [1] Apache lucene java.
<http://lucene.apache.org/java/docs>.
- [2] Lucene scoring. <http://lucene.apache.org/java/docs/scoring.html>.
- [3] Lucene's default similarity function.
<http://lucene.zones.apache.org:8080/hudson/job/Lucene-Nightly/javadoc/org/apache/lucene/search/DefaultSimilarity.html>.
- [4] Lucene's sweet spot similarity function.
<http://lucene.zones.apache.org:8080/hudson/job/Lucene-Nightly/javadoc/org/apache/lucene/misc/SweetSpotSimilarity.html>.
- [5] C. Buckley, A. Singhal, and M. Mitra. New retrieval approaches using smart: Trec 4. In *TREC*, 1995.
- [6] D. Carmel and E. Amitay. Juru at TREC 2006: TAAT versus DAAT in the Terabyte Track. In *Proceedings of the 15th Text REtrieval Conference (TREC2006)*. National Institute of Standards and Technology. NIST, 2006.
- [7] D. Carmel, E. Amitay, M. Herscovici, Y. S. Maarek, Y. Petruschka, and A. Soffer. Juru at TREC 10 - Experiments with Index Pruning. In *Proceeding of Tenth Text REtrieval Conference (TREC-10)*. National Institute of Standards and Technology. NIST, 2001.
- [8] O. Gospodnetic and E. Hatcher. *Lucene in Action*. Manning Publications Co., 2005.
- [9] S. E. Robertson, S. Walker, M. Hancock-Beaulieu, M. Gatford, and A. Payne. Okapi at trec-4. In *TREC*, 1995.
- [10] A. Singhal, C. Buckley, and M. Mitra. Pivoted document length normalization. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 21–29, 1996.